UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TREBALL FI DE FI DE GRAU

Grau en Enginyeria Electrònica Industrial i Automàtica

SISTEMA DE COMUNICACIONS ENTRE UN PC I UNA FPGA MITJANÇANT UN BUS DE DADES

Memòria i Annexos

Autor: Daniel Vicedo Contel
Director: Jordi Cosp Vilella

Convocatòria: Maig 2018

Resum

En aquest projecte es realitza el disseny, desenvolupament i test d'un sistema de comunicacions entre un PC i un dispositiu digital programable FPGA via un estàndard de comunicacions SPI. El sistema permetrà introduir les dades des d'una interfície gràfica d'usuari de l'ordinador per posteriorment transferir-les de la forma més eficaç possible al dispositiu programable i ser utilitzades per altres subsistemes connectats a aquest dispositiu.

El procés d'emmagatzematge de dades es porta a terme amb un mòdul VHDL el qual incorpora tot el codi de connexions SPI necessari per completar la transferència. La interfície gràfica que permet introduir les dades i controlar la configuració de la comunicació s'ha realitzat amb el programa Visual Studio.

Per demostrar el correcte funcionament del sistema dissenyat, es testejarà amb diferents ports de sortida de la FPGA per comprovar que el missatge s'ha enviat i enregistrat com es demana.

L'estructura es desenvoluparà perquè en un futur es pugui connectar a una entrada de dades d'una xarxa neuronal ja dissenyada.

Resumen

En este proyecto se realiza el diseño, desarrollo y testeo de un sistema de comunicaciones entre un PC y un dispositivo digital programable FPGA vía un estándar de comunicaciones SPI. El sistema permitirá introducir los datos des de una interface gráfica de usuario del ordenador para posteriormente transferirlas de la forma más eficaz posible al dispositivo programable y ser utilizadas por otros subsistemas conectados a este dispositivo.

Este proceso de registro se lleva a cabo con un módulo VHDL el cual incorpora todo el código de conexiones SPI necesario para completar la transferencia. La interface gráfica que permite introducir los datos y controlar la configuración de la comunicación se ha realizado con el programa Visual Studio.

Para demostrar el correcto funcionamiento del sistema diseñado, se testeará con distintos puertos de salida de la FPGA para comprobar que el mensaje se ha enviado y registrado como se pide.

La estructura se desarrollará para que en un futuro se pueda conectar a una entrada de datos de una red neuronal ya diseñada.

Abstract

This project implements the design, development and test of a communication system between a computer and a Field-Programmable gate array FPGA via SPI communication standard. The system will allow introducing data from a user graphic interface of the computer to be transferred afterwards as efficiently as possible to the programmable device and be used by other subsystems connected to the device.

The register process is done by a VHDL module which includes all the code for SPI connections that are needed to complete the transfer. The graphic interface that lets the user introduce data and control the configuration of the communication link is made by means of the program Visual Studio.

In order to demonstrate the proper functioning of the designed system, it will be tested with different outputs of the FPGA to check if the message has been sent and registered.

The structure will be developed so that in the future it can be connected to a data entry of an already designed neuronal net.

Agraïments

Primer de tot, he d'agrair al meu tutor, Jordi Cosp. Gràcies la seva ajuda i ànims al llarg del desenvolupament del projecte aquest projecte s'ha pogut dur a terme.

A Eva Barreda, la qual amb la seva companyia ha aconseguit fer les hores de biblioteca i laboratori molt més amenes i agradables.

Finalment, donar les gràcies als meus amics i familiars per donar-me tot el seu suport.

Glossari

Master: És l'encarregat de crear el senyal de rellotge i seleccionar l'esclau (*slave*) al que vol enviar o rebre la informació.

Slave: Rep o envia la informació quan s'ha seleccionat el seu senyal. No té capacitat per començar una comunicació per ell mateix.

Master Out Slave In (MOSI): Línia de comunicació on les dades surten del *master* i arriben al *slave.*

Master In Slave Out (MISO): Línia de comunicació contraria al MOSI; les dades surten del *slave* i les rep el *master.*

Slave Signal (SS): Senyal que permet seleccionar un *slave* en concret, és únic per a cada *slave* connectat al bus. També es pot anomenar Chip-Select (CS).

Register-transfer level (RTL): Disseny el qual modela els circuits digitals síncrons en termes de flux de senyals digitals entre els registres del hardware i les operacions lògiques.

Netlist: Descripció textual amb totes les connexions que hi ha formades en un circuit.

Floorplaning: És el procés de col·locar blocs en l'àrea del chip i determinar les rutes entre ells.

Índex

1. Prefaci

Aquest projecte parteix d'una idea prèviament establerta en un altre treball de fi de grau on s'hi introdueixen dades de manera manual a una xarxa neuronal. Tot i ser una manera molt robusta i segura d'enviar les dades correctament, limita en excés, la possibilitat de modificar les senyals segons com l'usuari decideixi. D'aquí sorgeix una necessitat de crear una interfície gràfica on l'usuari pugui enviar qualssevol senyal sense haver de modificar el codi intern o la placa on s'hi introdueix la xarxa neuronal. Amb aquesta idea s'han format les bases d'aquest projecte amb l'objectiu de suplir aquesta manca de versatilitat que es requereix per millorar la proposta de l'antic projecte de fi de grau.

L'estructura d'aquest treball es centrarà fonamentalment en la interfície gràfica i tots els aspectes que puguin mantenir la robustesa del projecte per a millorar la relació amb l'usuari i formar un enviament de dades més flexible.

Introduir les dades des d'una interfície gràfica implicarà crear una connexió extra amb la placa FPGA que abans no es requeria. S'hauran de buscar mètodes de recepció de dades que permetin aquesta connexió i que siguin el suficientment robusts com per continuar assegurant un enviament de dades adequat.

1.1 Requeriments previs

Plantejar un projecte de final de Grau partint de la idea d'un altre requereix informar-se no només de l'àmbit on es desenvolupa (fonaments, teoria implicada, software, hardware...) sinó que també s'ha d'estudiar la manera en com s'ha portat a terme i s'ha resolt. Així doncs, abans d'iniciar el procés de codificació del projecte s'investigaran diferents idees que es puguin incorporar dins d'aquest.

Els requeriments previs serien: plantejar com introduir les dades al codi ja completat, quines característiques han de tenir aquestes dades i formular una hipòtesis de quin protocol d'enviament de dades seria el més òptim per aquesta situació.

Alhora, per una altra banda, també s'haurà de fer una cerca envers als programes de creació d'interfícies gràfiques i decidir quin té més capacitats per mostrar el màxim d'opcions a l'usuari quan sol·liciti enviar les dades corresponents.

2 Introducció

En aquest projecte s'implementarà un codi que permetrà introduir una sèrie de dades a una FPGA que tingui la capacitat de processar-les i interactuar amb elles. Per dur a terme aquesta tasca es programarà una interfície gràfica senzilla i optimitzada que permeti a l'usuari introduir-hi les dades i enviar-les cap a la FPGA.

Ja que la interfície gràfica i el codi amb el que treballa la placa són escrits de diferents maneres i no es permet la connexió directe entre ells, es requerirà un mètode de comunicació que traslladi les dades d'una a l'altre. Per decidir quina és la millor forma de resoldre aquest conflicte, es realitzarà un estudi d'entre tots els protocols de connexió que hi ha actualment a la xarxa i s'escollirà el més adient per a la tasca en qüestió.

2.1 Abast del treball

El projecte s'estructurarà en tres fases principals i fonamentals:

Codificar una **interfície gràfica** que permeti introduir totes les dades necessàries per enviar correctament el missatge a la FPGA. També haurà de permetre la configuració de tots els paràmetres necessaris per a la connexió entre ambdues parts del projecte.

Programar un codi que s'utilitzi com a ***slave*** **de la connexió** el qual haurà de rebre les dades i les haurà d'emmagatzemar en registres interns que, posteriorment, s'exportaran a la implementació de la FPGA.

Per últim, s'haurà de crear una **connexió entre les dues etapes** ja esmentades ja que un codi VHDL no pot rebre dades directament d'una interfície gràfica. Així doncs, es formularà una connexió per enviar les dades d'una forma segura i llegible per ambdues parts de l'enviament.

Per fer més entenedora la explicació de com es connecten les dues parts entre elles, a continuació es mostra un esquema:

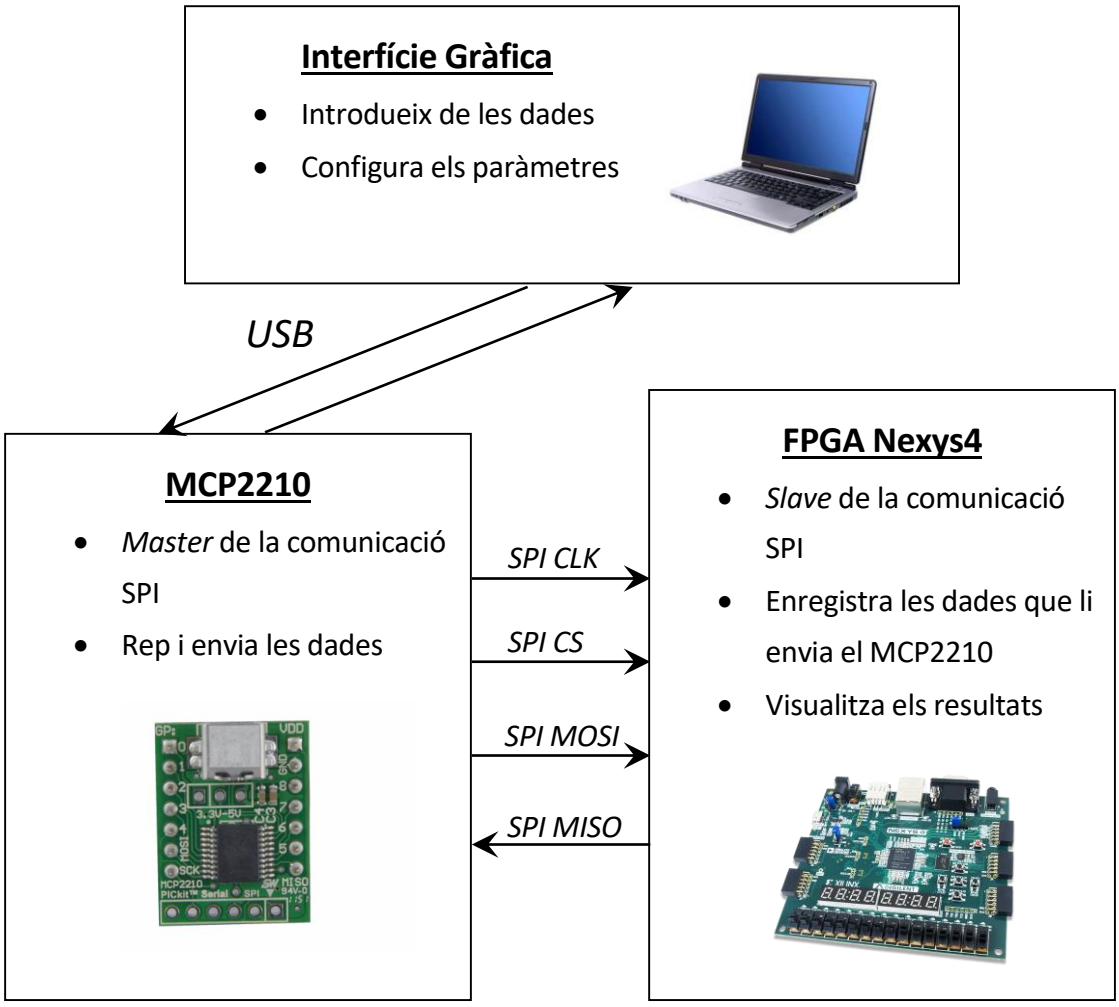

Figura 2.1. Esquema de les diferents parts del projecte.

Un cop finalitzat tot el procés de connexió entre ambdues parts del projecte s'iniciarà la última fase on es testejarà el funcionament i es comprovarà que totes les dades enviades des de la interfície gràfica arriben correctament a la FPGA.

2.2 Objectius del treball

A continuació es desglossen els objectius marcats pel projecte, que tenen com a finalitat permetre a l'observador del progrés al llarg de la implementació del codi, la interfície gràfica i els seus tests pertinents.

Els objectius són els següents:

- Recopilar informació referent als diferents tipus de estàndards de comunicació i decidir el més adient per les necessitats que es proposen pel projecte.
- Dissenyar el codi VHDL que permeti connectar el PC amb la FPGA.
- Adaptar el codi VHDL per permetre modificar imatges de dimensions 5x7.
- Programar una interfície gràfica eficient per configurar la connexió prèviament esmentada.
- Connectar el conjunt a una xarxa neuronal ja dissenyada.
- Augmentar la capacitat de modificació del codi per imatges de 28x28.

3 Marc teòric

3.1 Fonaments de la connexió SPI

La *Serial Peripheral Interfece* (SPI) és un protocol de comunicació síncron molt utilitzat per comunicacions en ambdues direccions entre dos dispositius. El bus SPI consta de quatre senyals:

- **CLK *(Clock Signal)*:** Aquesta senyal és el rellotge que s'utilitza per sincronitzar els bits de dades enviats i rebuts.
- **MOSI *(Master Out Slave In)*:** Senyal de sortida del *master SPI* per on s'envien les dades cap a l'*slave*.
- **MISO *(Master In Slave Out)*:** Senyal d'entrada que envia les dades de l'*slave* cap al *master*.
- **CS *(Chip Select)*:** Senyal que permet seleccionar la quantitat d'*slaves* actius per a l'enviament de dades. El *master* pot escollir quins *slaves* interactuen amb ell i se'ls pot enviar el missatge i quins es mantenen inactius.

Aquest protocol consisteix en un sol *master* i un o més *slaves* on el primer pot comunicar-se a qualsevol *slave* però aquests només poden fer-ho amb el *master*. Cada *slave* ha de tenir el seu propi senyal de selecció, el qual és únic i permet la identificació de cadascun d'ells, de tal manera que el *master* pugui escollir la senyal amb la que es disposi a comunicar-se. Com el SPI inclou una senyal de rellotge, l'únic requeriment perquè la comunicació es realitzi és que la freqüència d'aquest sigui inferior a la màxima freqüència de tots els dispositius connectats.

Quan el *master* del SPI vol iniciar una transferència, primer ha d'indicar a quin *slave* li vol transmetre i, seguidament introduir una senyal a nivell baix pel SS amb el que estigui relacionat l'*slave* en qüestió. Una vegada s'ha seleccionat, l'*slave* passa a escoltar la informació que es transmet pel bus i el *master* pot començar a fer la transferència de dades.

Hi ha quatre tipus de busos SPI relacionats amb les etapes del rellotge. Estan separats per dos paràmetres: CPOL i CPHA. CPOL o *Clock Polarity* relaciona la connexió amb l'estat de la senyal del rellotge (alt o baix). Per altra banda, CPHA o *Clock Phase* determina en quina banda de la senyal de *clock* s'envia la informació (flanc de pujada o baixada). Aquests paràmetres estan especificats en els full d'especificacions de tots els dispositius, així que es poden ajustar segons les necessitats de cadascun. El més comú és el CPOL = 0 (nivell baix) i CPHA = 0 (flanc de pujada). Durant el període on el senyal de selecció del dispositiu *slave* està activat, la informació es transmet entre els dos dispositius, un cop es desactiva, la connexió es finalitza.

En la imatge següent s'observen les diferents opcions que permet aquest protocol per transmetre les dades:

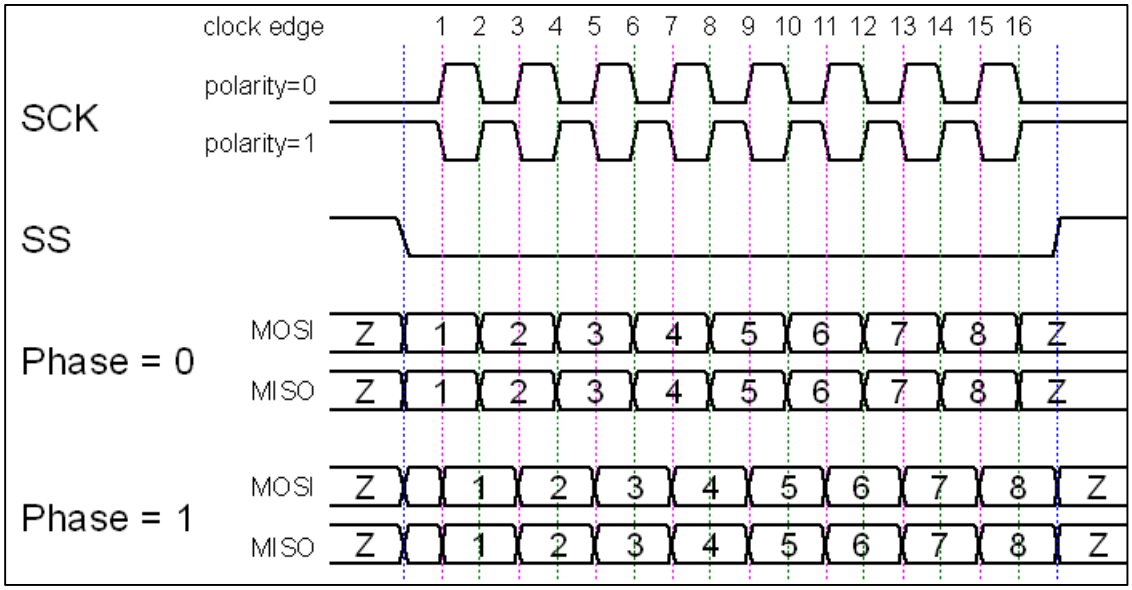

Figura 3.1. Diagrama amb els paràmetres CPOL, CPHA i SS. [1]

MOSI i MISO són pins els quals tenen les funcions de transmetre les dades al seu destinatari, en un cas és el *master* el que envia i l'*slave* el que rep (MOSI) i en l'altra el procediment és l'invers.

Qualsevol transmissió del protocol SPI està controlada únicament pel *master*. Aquest genera la senyal de rellotge i selecciona el senyal del *slave* amb el que vol comunicar-se. Això implica que els *slaves* no tenen la capacitat d'enviar informació cap al *master* per ells mateixos.

La comunicació entre els dos dispositius és simultània en ambdues direccions. D'aquesta manera, amb un ordre establert pel *master* s'envien i es reben les dades que es volen transferir. Aquest tipus de comunicació rep el nom de *full-duplex.*

3.2 Vivado

La finalitat d'aquest projecte es implementar el codi desenvolupat en un dispositiu programable en el qual es processarà dit codi segons convingui. Un dispositiu programable és un circuit integrat el qual està format per una matriu de portes lògiques que proporciona una solució al disseny digital d'una forma anàloga. D'aquesta manera es permet extreure les dades guardades en els registres interns i visualitzar-les en els ports externs que la placa porta incorporats.

El programa que s'utilitzarà per escriure el codi s'anomena Vivado i és de l'empresa Xilinx. Aquest software permet la síntesis i l'anàlisi temporal dels dissenys en VHDL per la posterior simulació a diferents estímuls provocats pel programador.

Vivado representa un gran avenç envers al seu predecessor ISE ja que gran part de les seves característiques no estaven implementades en el ISE, creat ara farà quinze anys. La major diferència entre aquests dos softwares és que Vivado porta incorporat un simulador lògic intern que permet realitzar simulacions molt més grans ja que no depèn d'un altre programa, com és el cas del ISE.

Les principals característiques que es poden trobar dins el programa Vivado són:

- Desenvolupament i anàlisi RTL *(Register-transfer level).*
- Configuració i implementació de IPs.
- Simulació lògica.
- Planificació de pins I/O.
- Síntesi lògica.
- Anàlisi de la *netlist* i Definició de restriccions.
- Anàlisi i *floorplanning* dels resultats de la implementació.
- Programació de dispositius, verificació i depuració.

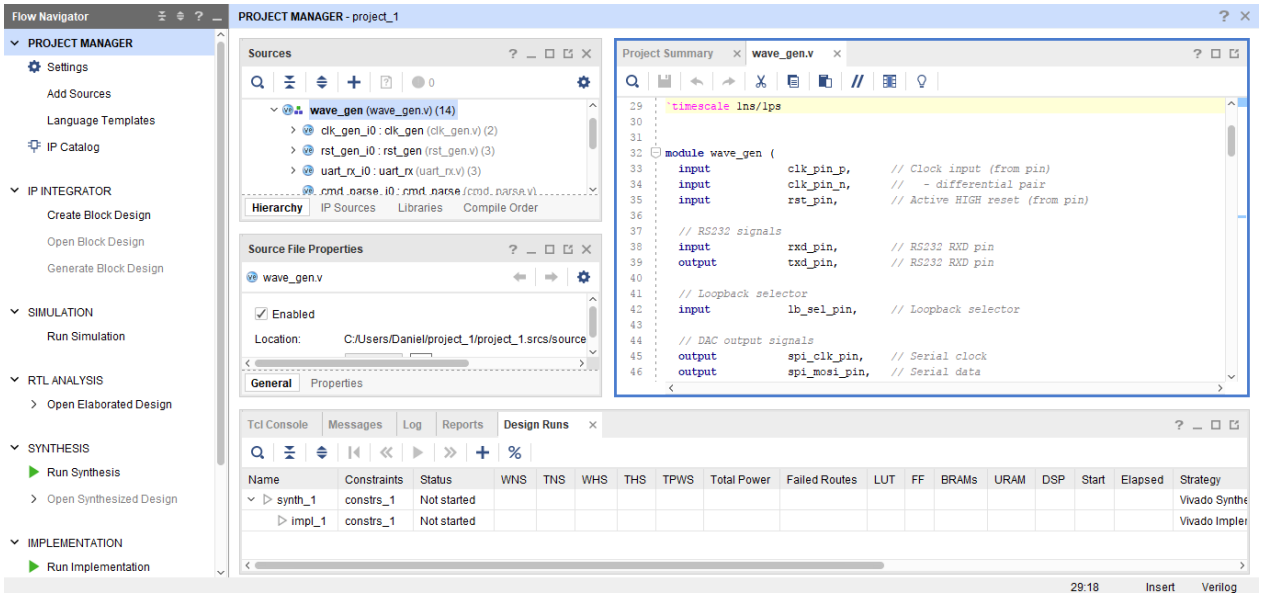

Figura 3.2 Estructura del software Vivado.

Per a que el programa Vivado pugui configurar-se correctament i formular les simulacions adequades, a l'hora de crear un projecte es mostren a l'usuari una sèrie de passos a seguir. Entre els

passos es troben: la selecció del nom del projecte, la possibilitat d'afegir els arxius que es vulguin carregar en el cas de que ja estiguin programats prèviament i també els arxius de *Constraints* que es requereixin per connectar els ports de la placa amb els senyals del programa. Finalment es demana que es seleccioni quina placa es farà servir quan s'implementi el programa al hardware per poder assegurar que els pins són els correctes i no s'envia cap senyal a un pin reservat per una funció concreta.

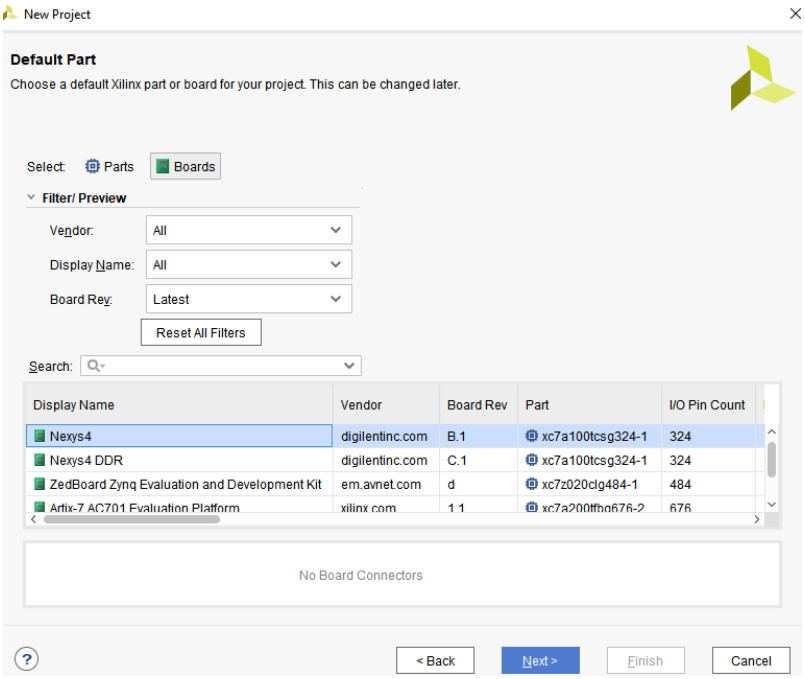

Figura 3.3 Selecció de la placa hardware del software Vivado.

Un cop el projecte ha estat configurat s'obren tots els documents que s'hi hagin inclòs i ja es pot iniciar la programació del codi en qüestió.

Per evitar possibles incidents i provocar un curtcircuit o un mal ús de la placa que s'estigui utilitzant, s'han d'assegurar el correcte funcionament tant de la síntesis del programa com de la implementació d'aquest mateix. Un cop s'han validat aquestes dues fases ja es pot crear un *Bitstream* que serà el que s'introduirà al hardware. Al llarg d'aquests processos de validació hi poden sorgir diferents tipus de missatges.

- **Estat:** Informa de cada pas que el software fa tant per la part de síntesis com per la d'implementació .
- **Informació:** Un cop acabat cada "estat" s'informa a l'usuari de que ha estat correctament realitzada la operació.

- **Advertència:** Són errors que no incapaciten la continuació de les fases de síntesis/implementació. Acostumen a ser senyals que no s'utilitzen.
- **Advertència Crítica:** Continuen permetent la síntesis o implementació però poden fer que el codi no funcioni correctament si no es revisen. L'ús de ports de la placa inconnexes en el codi poden ser un exemple d'advertència crítica.
- **Error:** En aquest punt el programa s'atura i sol·licita a l'usuari que investigui quin és l'error que s'ha produït i proposa uns suggeriments de com solucionar-lo.

3.3 Llenguatge VHDL

VHDL és un acrònim que combina VHSIC i HDL, on VHSIC es un acrònim de *Very High Speed Integrated Circuit* i HDL un altre de *Hardware Description Language*.

Aquest llenguatge s'utilitza per a la programació de FPGAs degut a que és un llenguatge paral·lel i no seqüencial com ho seria el C, per exemple. L'execució d'un programa en VHDL permet validar un disseny abans de la seva fabricació creant l'oportunitat de solucionar errors del codi abans d'implementar-lo a un dispositiu.

El més important dels llenguatges de descripció de hardware és que siguin capaços de simular perfectament el comportament lògic d'un circuit.

L'estructura de cada mòdul te dues parts diferents:

- **Entitat:** Aquí s'hi troben els paràmetres genèrics, les entrades i les sortides del mòdul.
- **Arquitectura:** Les diferents relacions entre mòduls, components i comportaments interns es troben en aquest apartat.

A continuació es mostra un exemple d'una porta NAND escrita i estructurada en VHDL:

```
-----------------------------------------------
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-----------------------------------------------
--Definició de la entitat de la porta NAND
ENTITY porta_NAND IS
      PORT    (a,b      :IN std_logic;
               Sortida :OUT std_logic);
END    porta_NAND;
-----------------------------------------------
--Definició de la arquitectura de la puerta NAND
ARCHITECTURE exemple OF porta_NAND IS
      BEGIN
            Sortida <= a NAND b;
END exemple
```

Figura 3.4. Exemple de codi en VHDL.

L'estructura del VHDL té un ordre concret que s'ha de seguir per a què compili correctament el codi. En el cas de que faltés una part necessària, a l'hora de sintetitzar i compilar el programa advertiria de quines parts manquen en la finestra d'errors i advertències.

Primerament, s'han d'especificar totes les llibreries que es requeriran al llarg del codi. Una llibreria és un conjunt de paquets de funcions que estan relacionades i poden agrupar-se. Aquests paquets també poden contenir informació de diferents "tipus" i "constants" que s'utilitzin freqüentment. La més utilitzada és la que es pot veure en la imatge anterior: **LIBRARY** IEEE.

Un cop escollides les llibreries, es poden seleccionar quins d'aquests paquets es requereixen. D'aquesta manera el codi pot anar a la ubicació de la funció que es vulgui utilitzar d'una manera molt més eficient que no pas buscant per tota la llibreria sencera. La més habitual és **USE** IEEE.STD_LOGIC_1164.ALL. En aquests paquets estan declarades les representacions lògiques dels valors en VHDL, així doncs, incorporant aquestes línies al codi, es permet seleccionar l'estat d'una variable en nivell alt, baix, no inicialitzat entre d'altres.

Seguidament, es declaren les entitats les quals serveixen per definir les entrades i sortides que tindrà un determinat circuit. Per definir una entitat s'utilitza la paraula reservada **ENTITY**. Dins l'entitat podem trobar dues instruccions diferents:

- **GENERIC:** Defineix i declara propietats o constants del mòdul. Per definir-les s'escriu el nom de la constant o propietat seguida de dos punts, el seu tipus i finalment s'introdueix el valor inicial amb un := prèviament.
- **PORT:** Defineix entrades i sortides del mòdul. Consisteix en indicar el nom de la senyal seguit de dos punts i la direcció del port. També s'ha d'indicar el tipus de senyal de la que es tracta. Els tipus de port poden ser:

- **IN:** Senyals d'entrada. Poden llegir però no se'ls pot assignar cap valor, és a dir, no es pot modificar el valor que posseeixen.
- **OUT:** Senyals de sortida. En aquest cas el seu valor si que pot ser modificat però no es pot llegir, per tant, no poden ser utilitzades per a l'assignació d'elements.
- **INOUT:** Aquest tipus és una mescla entre els dos anteriors i pot utilitzar-se tant com de lectura o d'escriptura.
- **BUFFER:** Idèntic a l'anterior amb la diferència que només una font pot modificar el seu valor.

Un cop s'acaba una entitat s'ha d'introduir la paraula reservada **END** seguida del nom del mòdul en qüestió.

Quan ja s'han inicialitzat totes les variables i mòduls, arriba el moment d'escriure l'arquitectura del codi. L'arquitectura és el que defineix com es comporta un circuit i s'ha de definir de la següent manera: **ARCHITECTURE** [nom] **OF** [nom de l'entitat a la que pertany] **IS**. Tot seguit s'introdueixen les instruccions per indicar la declaració de senyals, components o funcions de l'arquitectura. Aquests senyals són interns i no es pot accedir a ells des de l'entitat. A continuació, s'afegeix una línia amb la paraula reservada **BEGIN** que dóna pas a la descripció del circuit mitjançant una sèrie de sentències concurrents**.** Un exemple d'arquitectura seria el següent:

```
ARCHITECTURE exemple OF multiplexor IS
    SIGNAL sig1, sig2, sig3: BIT;
BEGIN
    sig1 <= NOT control;
    sig2 <= entrada1 AND sig1;
    sig3 <= entrada2 AND S;
    sortida <= sig2 OR sig3;
END multiplexor;
```

Figura 3.5. Exemple d'arquitectura en VHDL.

3.4 Visual Basic

Visual Basic és un programa de Microsoft el qual utilitza un llenguatge de programació basat en esdeveniments i permet crear un projecte de desenvolupament integrat. Permet que el programador, utilitzant components ja incorporats en el programa, creï una aplicació amb totes les capacitats per funcionar correctament.

Microsoft va crear Visual Basic amb la intenció de fer més visual la programació en C#. La idea es basa en programar a partir d'una base gràfica on es situen els components desitjats i clicant sobre d'aquests ja es forma l'estructura inicial en el propi codi. Per altra banda, el programa ajuda al programador proposant paraules clau en el moment que comença a escriure's una línia, de tal manera s'intenten evitar errades d'ortografia que impedirien al codi poder compilar correctament.

El motiu pel qual s'anomena llenguatge de programació basat en esdeveniments recau en com l'aplicació funciona. La persona que està utilitzant el programa interacciona amb ella i es crea l'estructura del codi automàticament. Hi ha diversos tipus de components que permeten la comunicació amb l'usuari com botons o quadres de text o *checkboxes*, per exemple. Tots aquests components pretenen ser el més intuïtius possibles per a que un usuari que no tingui relació amb el codi intern pugui interaccionar i fer funcionar correctament el programa.

3.4.1 Estructura Visual Basic

Dins de l'estructura del Visual Basic hi podem trobar:

- **Forms:** Són les finestres que es creen per la interfície de l'usuari.
- **Controls:** Components gràfics introduïts als *forms* per permetre a l'usuari interaccionar amb el programa.
- **Propietats:** Cada característica d'un *form* o control està especificada per una propietat. Alguns exemples de propietats serien el nom, la mida o el color. Visual Basic aplica una sèrie de propietats per defecte però és poden modificar en qualssevol moment.
- **Mètodes:** Procediments interns que poden ser invocats per produir una acció particular a un objecte concret.
- **Procediment d'Esdeveniments:** Codi relacionat amb algun objecte. Aquest codi ha de ser executat quan un esdeveniment concret succeeix.
- **Procediment Generals:** Codi NO relacionat amb els objectes. Aquest codi ha de ser executat per la pròpia aplicació i no per esdeveniment.
- **Mòduls:** Col·lecció de procediments generals, declaracions de variables i definició de constants utilitzades a l'aplicació

3.4.2 Procediments "Sub" i Funció

Els procediments anomenats "Sub" reben també el nom de subrutines. S'utilitzen quan hi ha tasques no relacionades amb objectes però requereixen d'una codificació. Aquestes subrutines ajuden a

dividir una aplicació complexa en diferents parts de codi més senzill d'estructurar i llegir. Un exemple de declaració de funció seria el següent:

```
Sub Nom_de_la_Subrutina(ByVal Argument As Tipus_de_l'argument)
        .
        .
End Sub
```

Aquestes subrutines permeten que dins el codi principal no hi hagi una gran quantitat d'informació secundària. Aquesta informació podria referir-se a accions com la de mantenir l'estructura del *form* o quines funcions tenen tots i cadascun dels objectes que pot observar l'usuari.

D'altra banda, els procediments Funcions realitzen accions dins el propi programa i retornen un valor. L'estructura d'una funció és la que es pot veure a continuació:

```
Function Nom_de_la_Funcio(Arguments) As Tipus_del_valor_a_retornar
        .
        .
End Function
```

Tota informació que es codifiqui dins de funcions o subrutines quedarà enregistrada únicament dins d'aquestes mateixes per defecte. En el cas que els resultats obtinguts volguessin traslladar-se a una altre part del codi, s'hauria d'especificar amb una definició **"Public"** al principi de la definició d'aquestes.

```
Public Function / Public Sub
```

Tant mateix, si es prefereix mantenir el valor de la variable només dins l'estructura de la funció/subrutina es pot incloure **"Private"** enlloc de **"Public"**. D'aquesta manera les variables quedarien internes i es podria repetir un mateix nom per diferents variables en diferents parts del codi. És freqüent utilitzar-ho per variables anomenades "contador" o "sumador" que s'utilitzen com a variables auxiliars només dins el codi d'aquestes funcions.

3.4.3 Variables

Les variables són utilitzades per mantenir informació necessària per l'aplicació. Les més freqüents són: **Boolean, Integer, String i Char**. Cadascun d'aquests tipus de variable té les seves pròpies propietats i s'han d'utilitzar acord amb els valors que s'hi vulguin introduir en elles.

Per exemple, si volem obtenir una resposta de sí o no, s'utilitzarà una variable **Boolean**, ja que només pot tenir valors iguals a "1" o "0". D'altra banda, si interessa guardar un valor numèric enter, es declararà una variable **Integer**. Aquest tipus de variable té diferents variants en funció de la longitud que composi el número enter convertit en bits i, si es requerís, també permet guardar nombres negatius. Les variables **Char** permeten guardar qualsevol tipus d'informació per visualitzar posteriorment però només poden guardar un símbol. Per a encadenar diferents Chars s'utilitza la declaració **String**, la qual permet ajuntar tot tipus de símbols, números i lletres i poder ser visualitzades al ser cridades pel codi.

Per declarar una variable, primer de tot s'ha d'introduir una paraula clau amb la que se li atorgaran unes propietats o altres. A continuació s'exposen les diferents paraules claus:

Es declaren amb la paraula clau **"Dim"** totes les variables que no mantenen el valor una vegada el procediment corresponent a acabat.

```
Dim MyInt as Integer

Dim MyB as Boolean
```

Si per altra banda, sí volem mantenir el valor un cop finalitza la part del codi, s'utilitzarà la paraula **"Static"**.

```
Static MyInt as Integer

Static MyB as Boolean
```

Finalment, es definirà una variable amb **"Global"** al davant si es pretén mantenir el valor resultant i alhora es vol utilitzar la mateixa variable en altres parts del codi. Acostumen a declarar-se al inici del codi per fer més senzilla la seva localització. Aquesta paraula clau faria el mateix servei que la paraula clau **"Public"** esmentat en l'apartat anterior de funcions i subrutines.

```
Global MyInt as Integer

Global MyB as Boolean
```

3.4.4 Objectes

Els objectes són la connexió entre l'usuari i el codi i tenen la funció de permetre la interacció entre aquests dos i s'han de codificar amb la intenció de fer tal interacció el més simple possible.

A continuació s'esmentaran els objectes de més freqüent ús:

- **Button:** S'utilitza per iniciar, interrompre o acabar un procés concret.
- **Label:** És un objecte sobre el qual es mostra un text prèviament escrit el qual l'usuari no té les capacitats de modificar. Es pot modificar durant la execució del programa si s'ha escrit en el codi. Pot mostrar resultats i anar-se modificant a mesura que l'usuari fa més esdeveniments.
- **Text Box:** Caixa de text que permet a l'usuari introduir dades.
- **Check Box:** Permet seleccionar-se per marcar l'activació del bit intern. Poden marcar-se múltiples Check Boxs alhora.
- **Option Button:** Similar al Check Box amb la diferència de que aquests són excloents. Només permeten la selecció d'un Option Button alhora. Serveix per marcar decisions com la selecció del sexe de l'usuari.

3.4.5 Classes

Les classes s'utilitzen per introduir diverses funcions i propietats, i poden ser cridades dins d'altres mòduls del codi principal. D'aquesta manera s'evita haver de repetir les declaracions de variables i/o funcions, i el codi queda molt més net i llegible.

En aquestes classes s'hi descriuen les regles que marquen el comportament d'un objecte, i estan dividides en interfície i estructura. La interfície descriu com es pot interaccionar amb la classe, mentre que la instància, a través de mètodes, descriu com es divideix la dada en camps més concrets.

Un exemple de classe seria el següent:

```
Class Nom {
        Membre_1;
        Funció_membre_1();
        Funcio_membre_2();
        Propietat_1
        Propietat_2
}
```

4 Disseny i anàlisi

El desenvolupament d'aquest projecte es pot dividir en tres etapes ben definides.

Primer de tot, la **programació de la interfície gràfica**. Aquest codi enviarà les dades que l'usuari sol·liciti en direcció al MCP2210.

Seguidament, el **MCP2210** i el **protocol d'enviament de dades SPI**. Aquesta placa MCP2210 es connectarà a l'ordinador via USB i farà de *master* en el procés de trasllat de dades entre l'ordinador i la FPGA.

Finalment, la **codificació d'un *slave* SPI** amb el llenguatge VHDL, que llegeixi i emmagatzemi les dades que enviï el MCP2210. Aquestes dades s'introduiran a una xarxa neuronal també codificada amb VHDL per a comprovar que el funcionament de tot el procés és el correcte.

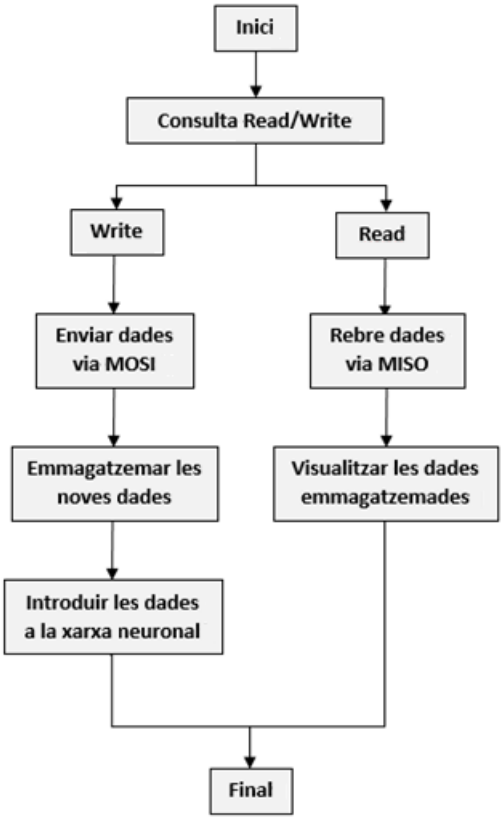

Figura 4.1. Diagrama dels processos del projecte.

4.1 Interfície gràfica

Amb el disseny de la interfície gràfica s'obtindrà la capacitat de decidir quins pins s'utilitzen per a la comunicació SPI alhora que permetrà editar i importar les dades que s'enviaran cap a la FPGA. A continuació s'esmenten les funcionalitats d'aquest programa:

- Configuració dels paràmetres necessaris pel correcte funcionament del MCP2210.
- Capacitat per sol·licitar quins pins s'activen per a l'enviament de les dades.
- Enviament de les dades cap al MCP2210 seguint el protocol SPI i el pin MOSI.
- Recepció de les dades enregistrades via el pin MISO del MCP2210.

4.1.1 Configuració del MCP2210

Pel correcte funcionament de la placa, s'han de configurar una sèrie de característiques prèvies a l'enviament de les dades. Aquestes característiques engloben els valors necessaris perquè les dades enviades siguin les correctes. Dins d'aquest conjunt es troben: El valor de la velocitat dels bits, els retards entre enviaments i les activacions dels GPIO "*General Purpose Input/Output*" conforme s'utilitzin o no.

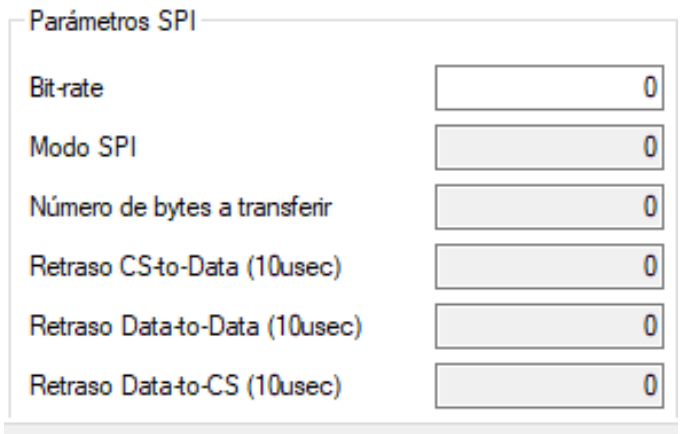

Figura 4.2. Pestanya per a la configuració del MCP2210.

A continuació s'esmenten els diferents paràmetres que es poden configurar per al MCP2210:

- **Bit Rate:** Nombre de bits per segon que es poden enviar per la xarxa digital.
- **Mode SPI:** El protocol SPI pot enviar les dades per flanc de pujada o baixada o també quan el valor és 1 o 0. En funció del que l'usuari necessiti s'introduirà un mode o un altre.

- **Número de bytes a transferir:** Nombre de bytes que es pretén enviar.
- **Retràs CS-Data:** Retràs que s'introdueix per evitar solapaments i glitch. El SPI s'espera un cert temps un cop rep l'activació del chip-select abans d'enviar la dada que pertoqui.
- **Retràs Data-Data:** En aquest cas, el retràs és entre bit i bit del missatge a enviar.
- **Retràs Data-CS:** Aquí s'inclou un retràs entre l'últim bit a enviar i la desactivació del CS.

En el cas que s'intentés fer una connexió SPI amb la configuració inicial (tots els valors a 0) apareixeria un missatge avisant que no és possible enviar dades amb 0 bytes de longitud i que s'ha de modificar el valor pertinent.

A causa de que les dades enviades en aquest projecte sempre són de la mateixa longitud, s'introduiran des de el Visual Basic i es bloquejarà el TextBox de tal manera que no es permeti a l'usuari modificar-lo ni crear errors en futurs enviaments.

4.1.2 Transferència de dades

Abans de poder comunicar l'ordinador amb el MCP2210, s'han de seleccionar quins pins es volen actius per a la transferència de les dades. En aquest projecte només se'n requereix un ja que només es fa servir un *slave* que rep tota la informació. Tot i així, s'ha de permetre a l'usuari escollir quin dels *slaves* vol activar i quins mantenir fora de la comunicació.

Quan la petició de l'usuari és d'enviar noves dades cap al MCP2210, la interfície gràfica ha de gestionar totes les configuracions pertinents per tal de que el pin MOSI detecti el nou missatge i el pugui enviar correctament cap a l'*slave* de la FPGA. D'altra banda, quan la sol·licitud és de lectura, és el pin MISO el que ha de rebre els registres guardats en la FPGA i la interfície gràfica ha de poder visualitzar-los en la pantalla.

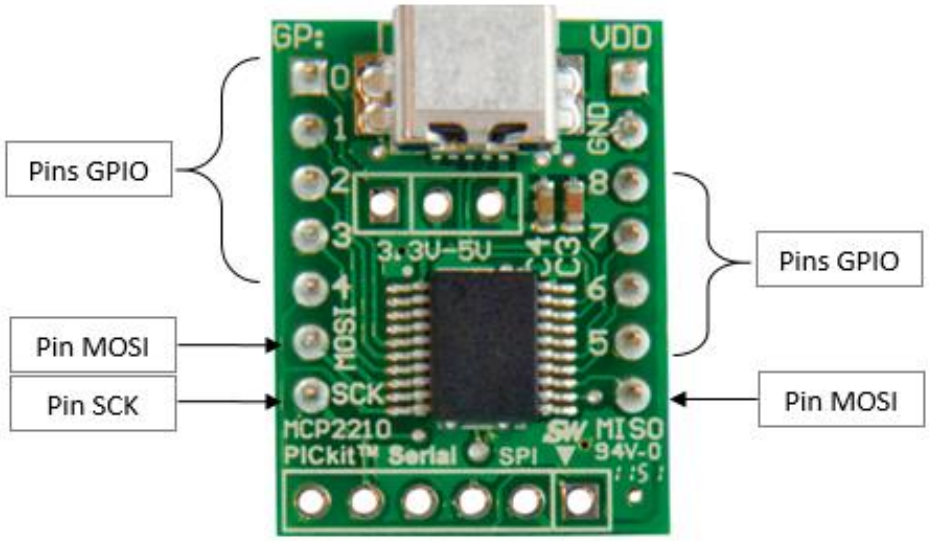

Figura 4.3. Disposició dels diferents pins del MCP2210.

Degut a que les dades que s'introdueixen en la interfície gràfica no es poden enviar directament perquè són del tipus *string* i no pas bytes com sol·licita el protocol SPI, per tal de permetre'n l'enviament s'hauran de transformar abans d'enviar-les. Per això, prèviament a la connexió SPI es modificarà el missatge de tal manera que sí sigui llegible pel MCP2210. En el cas de que s'intentessin enviar les dades en tipus *string* el MCP2210 no detectaria res i es mantindria a zero durant tot el procés d'enviament.

4.2 Comunicació SPI

Per a transferir les dades que ha introduït l'usuari a la interfície gràfica s'utilitza un tipus de comunicació anomenat "SPI" o "*Serial Peripheral Interface*". Aquesta comunicació permetrà l'enviament de les dades a la FPGA passant el pin MOSI de la placa MCP2210. Aquest procés es pot separar en dues fases ben contrastades, les quals són:

- Introduir les dades cap al bus SPI per el pin MOSI.
- Processar aquestes dades amb un dispositiu programable FPGA.

Un cop introduïdes les dades, el missatge es trenca en 35 bits individuals que s'enviaran un a un, tal i com el protocol SPI demana. Com només es poden enviar individualment, la comunicació es pot considerar en sèrie, cosa que facilita l'emmagatzematge posterior de les dades en els registres.

Com la connexió no pot ser directe perquè l'ordinador i la FPGA funcionen amb protocols diferents, s'implementa un pas entremig que transforma les dades d'un sistema a dades llegibles per l'altre. Aquest pas és possible gràcies a la placa MCP2210 de la marca Microchip.

4.2.1 MCP2210

El MCP2210 és un dispositiu convertidor USB-SPI Master que permet connectar aplicacions USB que tenen una interfície SPI. A continuació, es llisten una sèrie de característiques que incorpora la placa:

- Buffer de 128-byte dividits en 64-bytes per enviaments i 64-bytes per la rebuda de dades.
- Suporta els quatre tipus de modes SPI.
- Fins a 9 línies "*Chip Selects"* utilitzables amb qualssevol tipus de combinació per a les transferències SPI.
- EEPROM de 256-bytes.

L'objectiu d'aquesta placa és transformar les dades transmeses per l'ordinador per a que, posteriorment via el codi VHDL, sol·licitar-les i processar-les adequadament.

El MCP2210 funciona com el *master* de la comunicació SPI i és el que mana respecte a l'*slave* que s'introdueix al FPGA. D'aquesta manera, tot el que sol·liciti l'usuari via interfície gràfica haurà de passar pel *master* i mai hi haurà cap conflicte on es demanin varies operacions alhora que provoquin un mal funcionament del sistema.

Seguidament es mostra un diagrama de blocs de la placa on s'observen totes les parts que incorpora però cal esmentar que en aquest projecte només es centra en la part de la connexió SPI (color verd).

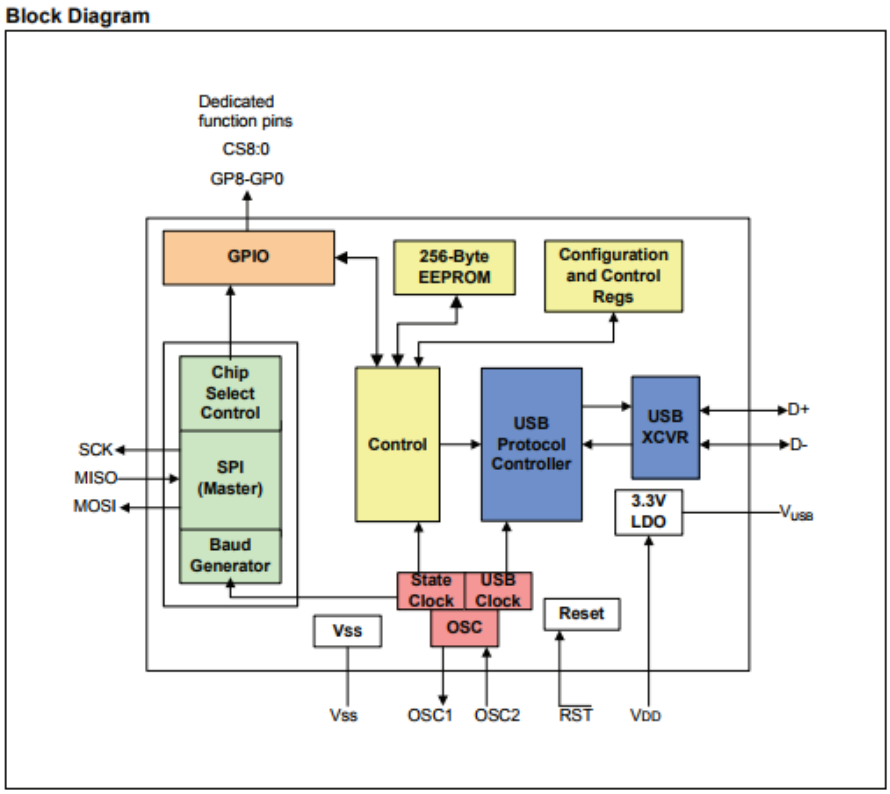

Figura 4.4. Diagrama de blocs de la placa MCP2210.[2]

Tota dada que es vulgui introduir al MCP2210 haurà de passar via USB ja que és l'única connexió que té la placa. El propi USB també serveix per donar el voltatge necessari per a què la placa es mantingui en funcionament.

4.2.2 Funcionament del MCP2210

La seqüència de passos que segueix el MCP2210 per a què la comunicació sigui correcta és molt senzilla de seguir i es pot trobar ben explicada al *Datasheet*.

Primer de tot, es configuren els paràmetres esmentats en l'apartat anterior: Valor de la velocitat d'enviament, longitud del missatge i retard entre les dades, com els GPIO que es tingui intenció d'utilitzar per la transmissió. Seguidament es creen una sèrie de polsos de rellotge per a la quantitat de bits a transmetre. D'aquestes seqüències n'hi pot haver de dos tipus: Les enviades pel MOSI o les rebudes pel MISO. En el cas de les rebudes s'introdueix el pas de convertir-les en dades llegibles per l'usuari des de l'ordinador, mentre que si es sol·licita un enviament de noves dades aquestes reescriuran les ja enregistrades a la FPGA.

Finalment es desactiven els GPIO i es queda a l'espera d'una nova transmissió.

A continuació, es mostra un ordinograma que explica les diferents etapes esmentades:

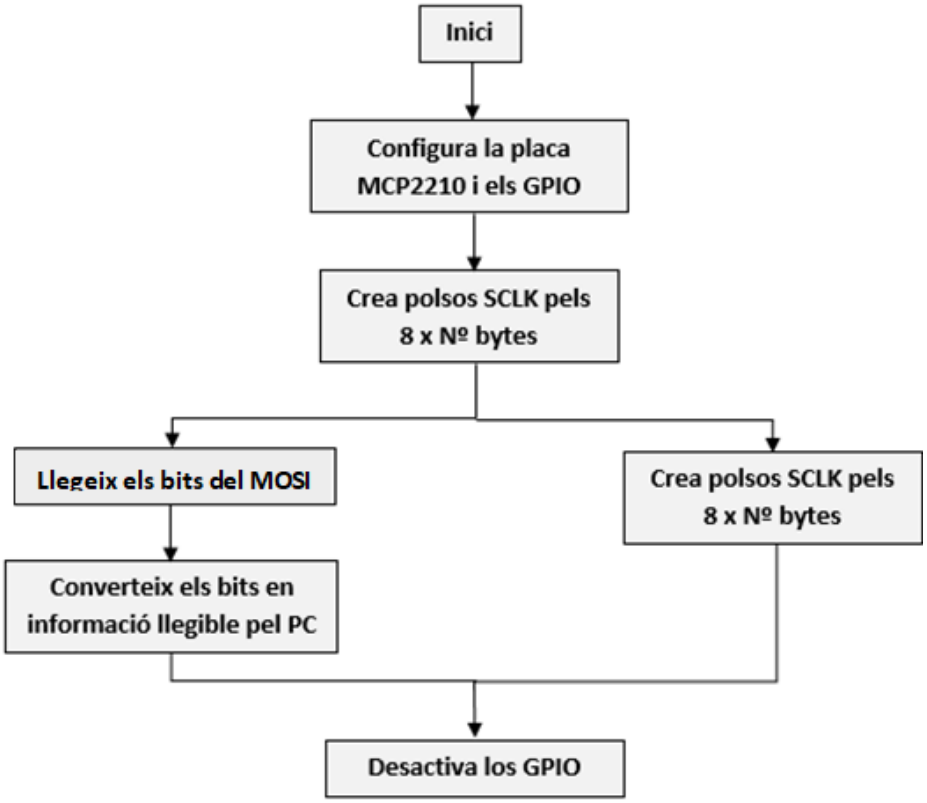

Figura 4.5. Ordinograma del funcionament de la placa MCP2210.

4.3 Mòdul VHDL

El mòdul VHDL serà l'*slave* en el protocol de comunicació SPI i, per tant, l'últim apartat del projecte. Les funcions que haurà de realitzar són les següents:

- Rebre les dades enviades des de l'ordinador i transmeses pel *master* SPI.
- Emmagatzemar les dades rebudes del MCP2210.
- Mostrar les dades en els ports de sortida del FPGA un cop tot el procés d'enviament i recepció de dades hagi conclòs.

El disseny s'agruparà en el mòdul "TOP" on s'hi introduiran tots els mòduls secundaris i s'hi instanciaran totes les variables necessàries. Aquestes variables són principalment les senyals que rep del MCP2210 i el seu propi *reset* per inicialitzar a zero tots els registres i senyals.

Dins el mòdul principal hi ha un mòdul intern anomenat “synchronizer” on s’hi crearan uns *buffers* per evitar solapaments entre senyals.

4.3.1 Configuració del missatge a enviar

Per assegurar que la transferència de dades es produeix correctament s’ha d’enviar el missatge amb uns paràmetres de control. Per aquest projecte s’ha dissenyat un protocol que permet enviar un missatge d’una longitud total de 16 bits. A continuació es desglossen les diferents parts del missatge:

- **Bit de *Read/Write*:** El primer bit que s’enviarà en cada transmissió serà per informar al *slave* de si ha d’enviar o rebre les dades. Quan aquest bit sigui ‘1’ s’entendrà com a *write* i, per tant, s’haurà d’iniciar el protocol d’enviament de dades des de el MOSI fins als registres. D’altra banda, quan sigui ‘0’ implicarà que es sol·licita *read* i s’hauran d’enviar els registres guardats via MISO cap al FPGA.
- **Adreça:** Els dos següents bits són per escollir en quina adreça es guardarà el missatge que s’està enviant. Com es pretenen guardar dades fins a tres registres diferents, quan el missatge comporti el valor “00” implicarà que les dades s’han d’introduir en el primer registre. En el cas de que sigui “01” s’emmagatzemaran en el segon registre i si l’adreça és “10” s’enviaran al tercer registre. D’aquesta manera s’assegura que un missatge no esborri l’altre al intentar enregistrar-los en la mateixa variable.
- **Primera part del missatge:** Els quatre següents bits seran la primera part del missatge que s’estigui enviant.
- **Zero de control:** Aquest zero s’introdueix per completar el primer byte (8 bits) del missatge complert. Permet que a continuació s’enviï la segona part del missatge sense que es solapi amb la primera.
- **Segona part del missatge:** Envia els 8 següents bits introduïts en la interfície gràfica.

Per fer més visual el missatge a transmetre es mostra un petit exemple a continuació:

Taula 4.1. Missatge a enviar desglossat per parts.

R/W	Adreça	Missatge	Zero	Missatge
1	00	1010	0	11110000
1	01	0011	0	01010100
1	10	1111	00	0001110

Així doncs, per aconseguir un total de 35 bits de missatge es requeriran 3 enviaments iguals al esmentat amb la diferència de que l'últim registre ha de tenir un bit menys de missatge. Per assegurar que s'envien sempre 16 bits i no es deixa res a l'aire, en el cas de l'últim registre s'enviaran dos zeros de control, obtenint així una major fiabilitat en l'enviament de les dades.

Seguint amb l'exemple anterior, el missatge que s'hauria introduït en la interfície gràfica seria "1010111100000011010101001111000111101", però per enviar-lo correctament es seguiran els passos esmentats anteriorment. Alhora, els registres on quedarien guardades les dades sol·licitades quedarien de la següent manera:

- Registre 0 = "101011110000"
- Registre 1 = "001101010100"
- Registre 2 = "11110001110"

Seguidament, es concatenaran els registres per a reconstruir el missatge que s'ha sol·licitat i poder visualitzar-lo posteriorment als ports externs de la FPGA.

4.3.2 Mòdul *slave* SPI

Aquesta part del codi té múltiples entrades i sortides, fonamentalment les necessàries per la connexió SPI. A continuació s'explica en detall cadascuna d'elles:

- **CLK:** Senyal de rellotge que permet el sincronisme de tot el sistema. Qualssevol acció es realitzarà quan aquest senyal variï d'una manera o una altra en funció de com estigui instanciat.
- **RST:** Senyal de reset el qual reinicia totes les variables i senyals del codi a zero.

- **SPI_CLK:** Senyal de rellotge molt més lent que el CLK del codi. Serveix per sincronitzar l'enviament de dades i que no es solapin entre elles durant la comunicació SPI.
- **SPI_CS:** Senyal de *chip-select* relacionat amb el SPI. Quan aquest senyal està activat es permet l'enviament de dades.
- **SPI_MOSI:** Senyal *Master Output Slave Input.* Bit que s'envia des del MCP2210 fins al codi. Com l'enviament és en sèrie el senyal només ha de guardar un bit a cada senyal de SPI_CLK.
- **SPI_MISO:** Senyal *Master Input Slave Output.* Mateix funcionament que l'anterior senyal però en aquest cas és el codi VHDL qui envia les dades enregistrades cap al MCP2210.

D'altra banda també s'incorporen diferents senyals internes les quals serveixen per guardar resultats de diferents operacions. Seguidament s'esmenten dits senyals interns:

- **REG0, REG1, REG2:** Registres on es guarden les dades rebudes pel MOSI. Posteriorment es concatenen per obtenir el missatge complert el qual serà el que s'enviarà a mostrar.
- **WRITE_OUT:** Registre on es guarda el missatge ja concatenat. Es guardaran les dades en aquesta senyal quan la sol·licitud del MCP2210 sigui de sobreescriure els registres amb noves dades.

A continuació es mostra un ordinograma amb les diferents funcions que pot proporcionar l'*slave* codificat en VHDL. Sempre s'ha de tenir en compte que es requereix una sol·licitud de part del *master* de lectura o escriptura per a que el codi actuï i que mai actuarà per iniciativa pròpia.

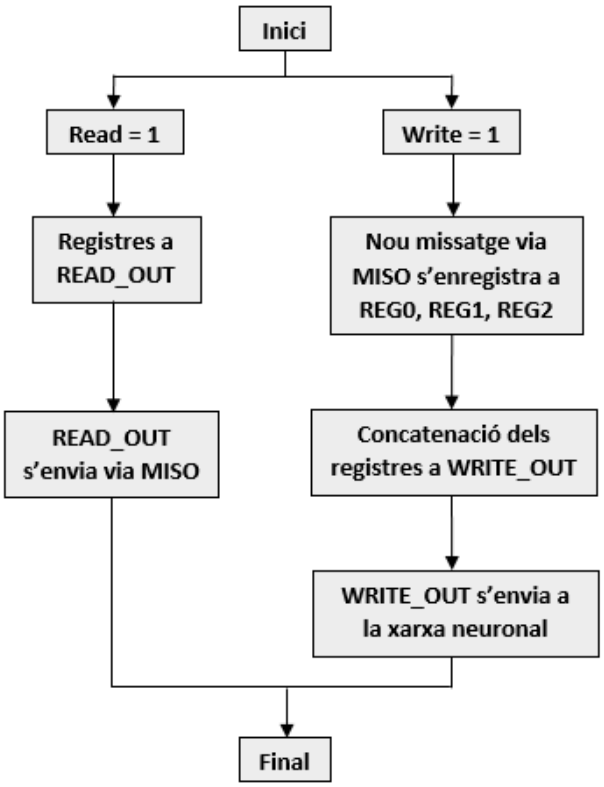

Figura 4.6. Diagrama funcional del mòdul *slave* SPI.

4.3.3 Màquina d'estats

A l'interior de l'*slave* s'hi troba una màquina d'estats que, segons l'estat en el que es trobi, permetrà l'enviament de les dades i l'enregistrament d'aquestes als registres. Aquest procés serveix per evitar una comunicació contínua amb el MCP2210 i que només s'enviïn les dades quan el missatge estigui complert. Els diferents estats del procés s'anomenen:

- **Idle:** Estat d'espera. Mentre no s'estiguin reben dades es mantindrà aquest estat de pausa per evitar la comunicació constant amb el *master.*
- **RX:** Estat de recepció de dades. Quan el *chip-select* s'activa, la màquina d'estats es mou a aquest estat per començar a rebre les dades del *master*.
- **Transfer:** Última fase de la màquina d'estats on es transmet el missatge complert al registre que pertoqui. Només dura l'instant que la transferència tarda en enregistrar el missatge, un cop acabada es retorna a l'estat Idle.

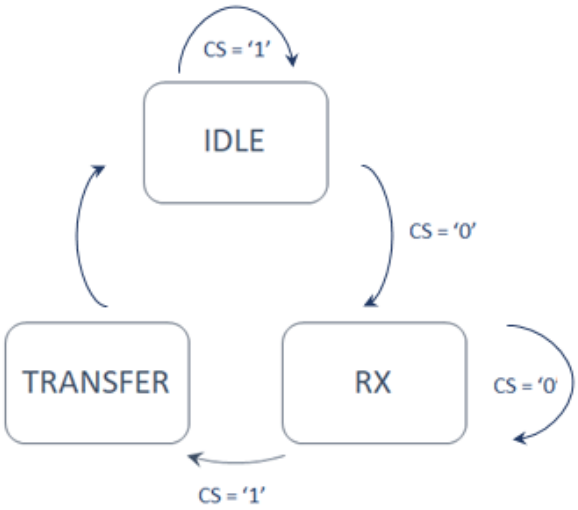

Figura 4.7. Diagrama d'estats de l'*slave* SPI.

Així doncs, gràcies a les diferents senyals de control i sincronisme, amb aquest programa es poden fer funcionar els processos quan se'ls sol·licita correctament.

Les principals senyals de control esmentades són el rellotge (CLK), el *Reset* (RST), el *chip-select* (SPI_CS), l'estat en el que es troba la màquina d'estats i el primer bit del missatge (SPI *Read/Write* bit). Per tant, aquestes són les senyals necessàries per a que els processos s'iniciïn.

En el cas de que el *Reset* estigui actiu, totes les senyals es reinicien al seu valor inicial:

- Registres del sincronitzador = '0'
- Registres d'emmagatzematge = '0'
- Registres concatenats per enviar = '0'
- Màquina d'estats = Idle

Per altra banda, mantenint el *Reset* desactivat, s'activa el *chip-select* aleshores:

- Canviarà l'estat de la màquina a RX per iniciar la transferència de dades.
- Es mantindrà aquest procés fins que es desactivi novament el *chip-select*.

4.3.4 Mòdul sincronismes SPI

Dins aquest mòdul es troben els diferents *buffers* que permeten una correcte transferència de dades per la comunicació SPI alhora que atorguen un petit marge de temps al codi VHDL per emmagatzemar-les en els registres pertinents. Les senyals que s'utilitzen en aquest codi són les següents:

- **CLK:** Mateix senyal de sincronisme que en el mòdul principal. Serveix per evitar el solapament en l'enviament de dades.
- **RST:** Reinicia tots els registres a zero. Mateix senyal que al codi principal.
- **IN_SIG:** Senyal d'entrada al *buffer*.
- **OUT_SIG:** Senyal de sortida del *buffer.*

L'estructura del *buffer* és senzilla, només ha de transferir una dada d'entrada a una de sortida sense modificar-la. Aquest procés es un tràmit que s'utilitza per a que la resta del codi principal pugui realitzar totes les seves funcions sense que li entrin dades a massa velocitat i es pugui produir un error en l'emmagatzematge de les dades rebudes.

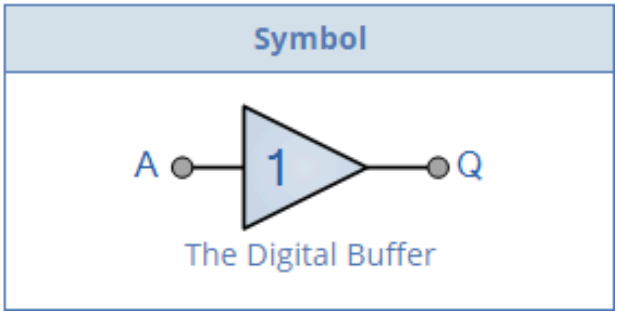

Figura 4.8. Símbol del *buffer* digital. [7]

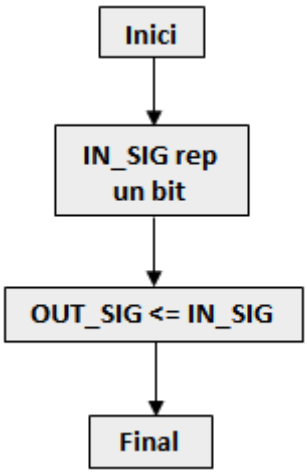

Figura 4.9. Diagrama funcional del *buffer* digital.

4.3.5 Mòdul TOP

Aquest mòdul no té cap funcionalitat més que la d'englobar tots els mòduls ja esmentats prèviament. D'aquesta manera totes les dades que vinguin de l'exterior hauran de passar per aquest mòdul abans

d'introduir-se en els diferents apartats del programa. També s'aprofita aquest mòdul per assegurar el sincronisme en tot el projecte amb les senyals de CLK i RST.

Totes les connexions del programa quedarien configurades de las següent manera:

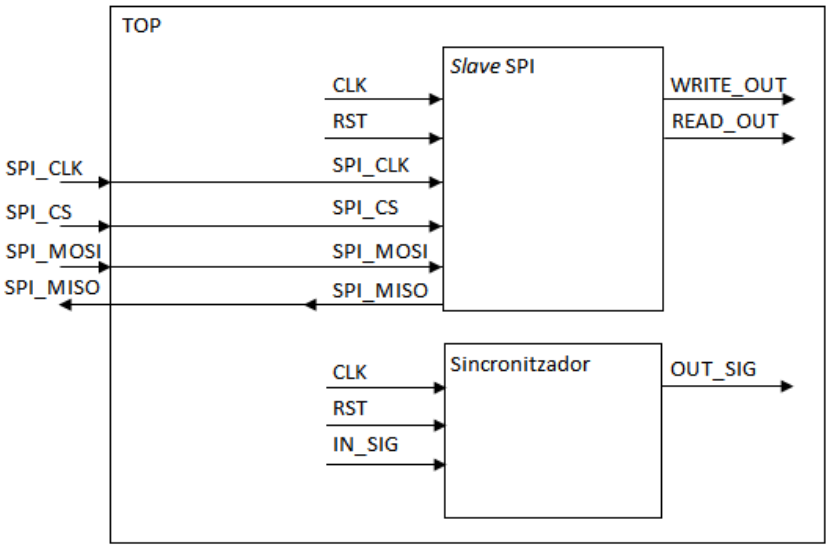

Figura 4.10. Esquema intern del mòdul TOP.

Els senyals que no surten del quadre TOP són senyals interns que no tenen cap funció externa al codi. Aquests senyals obtenen valors gràcies a processos interns que comporten la comunicació entre el mòdul de sincronització i l'*slave* o entre l'*slave* i la sortida del sistema.

5 Desenvolupament dels mòduls VHDL

Els mòduls es desenvoluparan tal i com s'ha mostrat en el disseny en anteriors capítols. Estarà tot englobat en el mòdul "Slave SPI" i l'arxiu XDC on es troben totes les connexions d'entrada i sortida que s'utilitzaran per testejar el codi amb la FPGA.

Primer de tot, es definiran cadascun dels mòduls que es dissenyaran i, en el cas de l'*slave* de l'SPI, també es dissenyarà un codi de simulació per assegurar que els senyals de sortida canviïn quan se'ls demana i d'aquesta manera no es produeixi cap error a l'hora d'introduir el codi a la FPGA.

Cada mòdul es simularà per mostrar visualment com funciona i fer més senzilla l'explicació en profund de cada part del codi.

5.1 Slave SPI

Tal i com s'ha esmentat en el capítol de disseny, l'*slave* de l'SPI s'hi inclouen totes les parts necessàries per a que el codi rebi les dades que se li enviïn, les emmagatzemi als registres pertinents i, finalment, les mostri als ports externs de la placa quan s'hagin verificat totes les dades rebudes.

5.1.1 Recepció de les dades

La recepció de dades és el primer pas de l'*slave* per a obtenir el missatge i poder enregistrar-lo i enviar-lo quan se li sol·liciti. Aquest procés s'inicia quan hi ha un flanc de baixada en els polsos de rellotge del SPI. Per comprovar que hi ha un pols de baixada es crearà una variable anomenada "flanc_sclk" que s'activi quan hi hagi una diferència de valor entre el rellotge del SPI "sclk" i el valor en l'anterior pols de rellotge SPI, "pre_sclk". D'altra banda, només s'activarà "flanc_sclk" quan el *chip-select* estigui activat per assegurar que l'enviament s'està portant a terme quan hi ha un missatge a enviar i no en cap altre moment on s'enviarien bytes buits.

```
process (clk)
begin
    if (clk='1' and clk'event) then
        if (rst='1') then
            pre_sclk <= '0';
        else
            pre_sclk <= sclk;
        end if;
    end if;
end process;
-- activem flanc quan flanc de pujada de sclk i chip select (cs)
flanc_sclk <= '1' when (sclk='1' and pre_sclk='0' and cs='0') else '0';
-- carrega entrada serie (MOSI) a registre de sortida de 16 bits
--   fem sevir el senyal 'registre' com a vector temporal ja que al ser 'reg_out'
--   un vector de sortida no el podem llegir
process (clk)
begin
    if (clk='1' and clk'event) then
        if (rst = '1') then
            registre <= (others=>'0');
        elsif (flanc_sclk = '1') then
            registre(15 downto 1) <= registre(14 downto 0);
            registre(0) <= mosi;
        end if;
    end if;
end process;
```

Figura 5.1. Codi de recepció de dades de l'*slave* SPI

Aquesta part del codi té múltiples entrades i sortides, les quals s'explicaran després de la simulació que es mostra a continuació:

- Període del rellotge: 10 ns.
- Període del CLK_SPI: 50 ns.

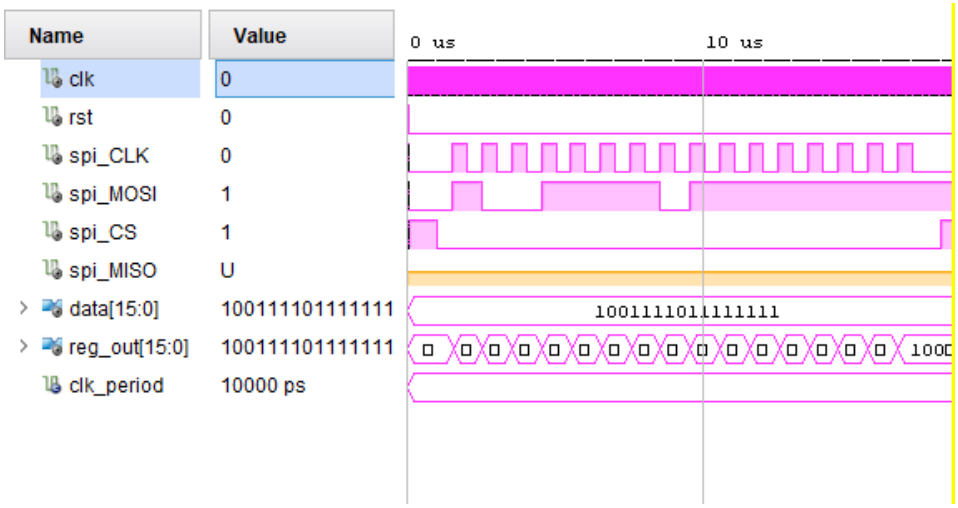

Figura 5.2. Simulació del mòdul *slave* enviant informació.

En aquesta simulació es mostren totes les variables de les que en fa ús l'*slave* de l'SPI per rebre les dades que se li envien. Com es pot comprovar en el moment en que el "spi_CS" s'activa a nivell baix comença a funcionar el "spi_CLK" i en conseqüència, la transmissió de dades. En cada senyal de rellotge del SPI es transmet una dada individual extreta del senyal "data" el qual és el missatge que se li ha introduït a la interfície gràfica i es pretén emmagatzemar als registres posteriorment.

El senyal "reg_out" és una variable auxiliar que permet guardar les dades individuals que s'envien pel "spi_MOSI" D'aquesta manera s'assegura que la connexió que s'està portant a terme és en sèrie i no es transmeten més dades de les que el protocol SPI pot llegir alhora.

D'altra banda, també es mostren els estats de la màquina d'estats que permet la transmissió de les dades rebudes cap als registres. En el moment en què el *chip-select* s'activa a zero, l'estat varia a "rx" on es comencen a enviar les dades via el MOSI i en el moment que es desactiva el *chip-select* és quan la màquina d'estat torna a canviar a un estat anomenat "transfer" que permet l'emmagatzematge de les dades.

5.1.2 Enregistrament de les dades

En els registres es guarden les diferents parts del missatge complert, d'aquesta manera les dades no es perden al llarg de la comunicació i es poden recuperar quan es requereixin. Així doncs, l'enregistrament es durà a terme cada cop que s'enviïn 16bits d'informació del missatge, ja que és la longitud màxima que el protocol SPI pot enviar alhora sense crear cap *glitch* pel mig.

```
register_process: process(clk, rst)
begin
     if (clk='1' and clk'event) then
        if (rst = '1') then
            reg0 <= (others=>'0');
            reg1 <= (others=>'0');
            reg2 <= (others=>'0');
        elsif (state = transfer) and (registre(15) = '1') then  --registre(15) = '1' = write
            case (registre(14 downto 13)) is
                when "00" => reg0 <= registre(12 downto 9) & registre(7 downto 0);
                when "01" => reg1 <= registre(12 downto 9) & registre(7 downto 0);
                when others => reg2 <= registre(12 downto 9) & registre(6 downto 0);
            end case;
        end if;
    end if;
end process;
```

Figura 5.3. Procés d'enregistrament de l'*slave* SPI.

Aquest procés està sempre sincronitzat amb els polsos de rellotge "CLK" i el *reset* "RST" per assegurar que les dades arriben una a una i no es solapen entre elles.

Les variables que s'utilitzen en l'enregistrament són les següents:

- **Reg0, Reg1, Reg2:** En aquestes variables s'introduiran les diferents parts del missatge que s'hagi d'emmagatzemar.
- **Registre:** Variable interna amb la funció de guardar les dades que s'estan rebent per després introduir-les en els registre que pertoqui. Aquí s'hi introdueix tot el missatge i després es desglossa en diferents parts que seran explicades més endavant.

Com tot procés síncron, quan el senyal de "RST" està actiu, tots els registres es retornen al seu valor inicial, en aquest cas a zero. Un cop es desactiva el senyal de *reset* s'ha de verificar que el missatge que s'està rebent és correcte, i és per això que no s'enregistrarà res a les variables a menys que la sol·licitud que envia la interfície gràfica sigui de escriptura.

Per comprovar que el missatge és d'escriptura i ha de sobreescriure els registres emmagatzemats es valorarà el primer bit de cada enviament.

```
elsif (state = transfer) and (registre(15) = '1') then  --registre(15) = '1' = write
```

Figura 5.4. Comprovació del primer bit del missatge.

Si el senyal "registre(15) = '1'" és confirma, implicarà que el missatge és d'escriptura i es podrà iniciar el procés d'emmagatzematge. En el cas de que fos igual a zero, implicaria que la sol·licitud ha estat de lectura i per tant els registres no haurien de canviar de valor durant aquell procés.

Seguidament, s'ha de seleccionar un dels registres on s'introduiran els següents bits que arribin via MOSI. Per escollir quin registre guarda els bits, s'estructura un *case … is* amb les diferents opcions que hi pot haver:

```
case (registre(14 downto 13)) is
    when "00" => reg0 <= registre(12 downto 9) & registre(7 downto 0);
    when "01" => reg1 <= registre(12 downto 9) & registre(7 downto 0);
    when others => reg2 <= registre(12 downto 9) & registre(6 downto 0);
end case;
```

Figura 5.5. *Case…is* del segon i tercer bit del missatge.

En aquesta part del codi el que es comprova són el segon i tercer bit del missatge que s'està enviant des del *master*. Segons el valor que tingui, s'enregistrarà en una variable o en una altra aconseguint d'aquesta manera evitar solapaments i mantenir tot el missatge intacte.

- Si el valor és equivalent a **"00"** s'enregistraran els valors en la variable **"reg0"**.
- Si el valor és equivalent a **"01"** s'enregistraran els valors en la variable **"reg1"**.
- Si el valor és equivalent a **"10" o "11"** s'enregistraran els valors en la variable **"reg2"**.

El que s'introdueix en els registres no és tot el missatge, ja que hi ha unes condicions d'enviament que no permeten enviar totes les dades directament i, per tant, s'han d'escollir les dades correctes per guardar-les al registre.

Com ja s'ha comentat en l'apartat de disseny, el missatge complert que s'envia per part de la interfície gràfica és el següent:

Taula 5.1. Exemple del missatge a enviar desglossat per parts.

R/W	Adreça	Missatge	Zero	Missatge	Registre
1	00	1010	0	11110000	Reg0
1	01	0011	0	01010100	Reg1
1	10	1111	00	0001110	Reg2

De totes les dades que s'envien només interessa guardar les parts del missatge, és per això que només es guarden les que es troben en: "registre(12 downto 9) & registre(7 downto 0)". En el reg2 s'utilitzen dos zeros de control per assegurar que la longitud de 35 bits del missatge s'envia correctament. Si no s'afegís aquest zero extra, podria comportar un error a l'hora d'enviar l'última part del missatge perquè un bit quedaria enlaire.

Finalment, quan els registres ja s'han guardat correctament, es concatenen entre ells per aconseguir la cadena de bits final. Això es realitza al final del codi per assegurar que s'han seguit tots i cadascun dels processos anteriorment esmentats abans d'ajuntar els registres. D'aquesta manera s'evita enviar dades errònies o de sol·licituds anteriors.

```
write_out <= reg0 & reg1 & reg2;
```

Figura 5.6. Concatenació dels registres.

Per entendre millor el que s'ha explicat, seguidament es mostra una simulació amb uns valors que permetin una lectura senzilla de cadascun dels registres i de com es concatenen finalment entre ells:

- Missatge: "11111111111100000000000010101010101"

- Període del rellotge: 10 ns.
- Període del CLK_SPI: 50 ns.

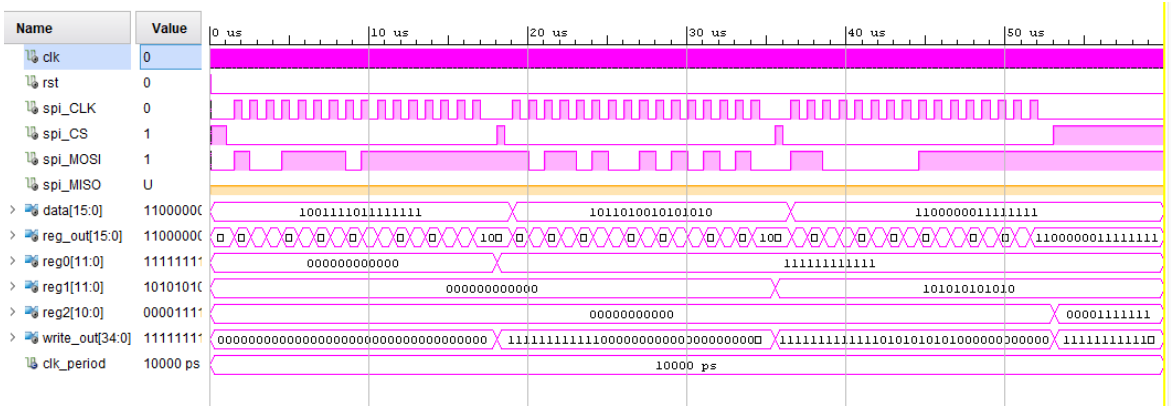

Figura 5.7. Simulació de l'*slave* SPI.

Com es pot veure en la simulació, cada cop que una transferència de 16 bits es completa, el missatge enviat es guarda en el registre corresponent. En aquest cas concret:

- **"reg0"** queda amb el valor de **"111111111111"**.
- **"reg1"** queda amb el valor de **"000000000000"**.
- **"reg2"** queda amb el valor de **"10101010101"**.
- **"write_out"** queda amb el valor de **"**111111111111000000000000010101010101**"**.

Tots els registres es mantenen amb el seu valor fins que no hi ha un nou estat de transferència en la màquina d'estats, la qual s'explicarà més detalladament en el següent apartat.

5.1.3 Màquina d'estats

La màquina d'estats que incorpora aquest codi permet l'inici de la seqüència de transmissió de dades cap a l'*slave* SPI. És necessària per evitar que hi hagi transmissions quan no s'han sol·licitat les quals puguin provocar un error en l'enviament o l'emmagatzematge de dades. Impedeix també qualsevol interrupció que pugui ocórrer mentre s'estigui portant a terme la transferència del missatge.

```
maquina_estats: process(clk, rst)
begin
     if (clk='1' and clk'event) then
        if (rst = '1') then
            state <= idle;
        else
            case state is
                when idle =>
                    if spi_CS = '1' then
                        state <= idle;
                    else
                        state <= rx;
                    end if;
                when rx =>
                    if spi_CS = '0' then
                        state <= rx;
                    else
                        state <= transfer;
                    end if;
                when transfer =>
                    state <= idle;
                when others =>
                    state <= idle;
            end case;
        end if;
    end if;
```

Figura 5.8. Màquina d'estats de l'*slave* SPI.

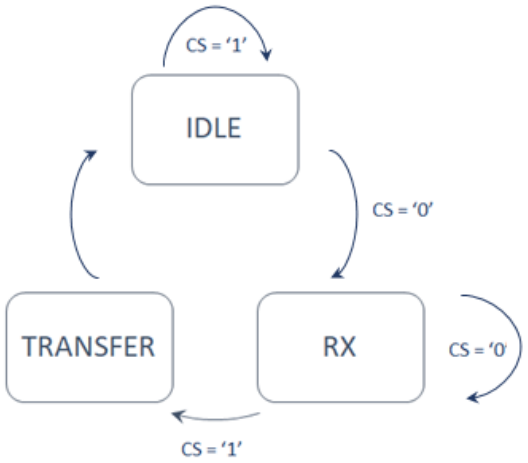

Figura 5.8b. Diagrama d'estats de la màquina.

Cada estat proporciona unes funcions necessàries per a la transferència òptima de les dades. La major part del temps, s'hi trobarà en l'estat de repòs o IDLE. Aquest estat es manté mentre no hi hagi cap sol·licitud de transferència. D'altra banda, també s'activarà l'estat IDLE quan es premi el botó de *reset* i s'inicialitzi tot el codi.

Per saber si s'inicia una transferència el codi comprova constantment el senyal de *chip-select* el qual s'activa en el precís instant en el que les dades comencen a enviar-se. Així doncs, quan *chip-select* s'activa, la màquina d'estats canvia a un nou estat.

Seguidament, a l'estat de RX o recepció de dades. L'estat es mantindrà actiu al llarg de la durada de l'enviament de dades i s'aniran carregant una a una al senyal interna de registre prèviament esmentat. Un cop s'hagin transmès els 16 bits que pertoquen el *chip-select* es desactivarà fent canviar de nou la màquina d'estats.

Finalment, l'estat de Transfer o traduït "Transferència" s'activarà només per un pols de rellotge, el qual és suficient per introduir totes les dades obtingudes al registre que pertoqui. Quan s'hagin introduït les dades al registre, la màquina d'estats tornarà al seu estat inicial amb la intenció d'esperar al pròxim enviament de dades per part del *master* SPI.

5.2 Sincronitzadors

Els sincronitzadors tenen la finalitat de proporcionar un petit període de temps entre que la dada es rep fins que s'incorpora al codi principal. Serveixen per evitar estats metastables i, d'aquesta manera evitar que els bits no es solapin entre ells a l'hora de l'enviament de les dades. La metastabilitat és la capacitat d'un sistema digital per mantenir una senyal inestable i no ser capaç de fixar-la en 1 o 0.

Aquests sincronitzadors utilitzen l'estructura d'un *buffer* la qual permet processar correctament dada per dada el missatge que s'està rebent

L'estructura seguida per inicialitzar aquestes variables és la següent:

```
entity synchronizer is
    generic (longitud : integer := 2);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           in_sig : in STD_LOGIC;
           out_sig : out STD_LOGIC);
end synchronizer;
```

Figura 5.9. Entitat dels sincronitzadors.

La variable "longitud" és una constant que permet escollir la quantitat de bits que hi caben a l'interior dels senyals d'entrada i sortida del *buffer*.

Un cop inicialitzades les variables es codifica el procés que es durà a terme dins aquesta part del codi.

```
process(clk)
begin
    if (clk='1' and clk'event) then
        if (rst = '1') then
            registre <= (others => '0');
        else
            registre(longitud-1 downto 1) <= registre(longitud-2 downto 0);
            registre(0)<=in_sig;
        end if;
    end if;
end process;
```

Figura 5.10. Codi del procés dels sincronitzadors.

Aquest procés, com els altres ja comentats, és síncron amb els polsos del rellotge CLK. Alhora també es fa ús del *reset* i de la seva funció de reiniciar totes les dades al seu valor original, en aquest cas la variable "registre" és la que es retornaria a valor zero.

D'altra banda, un cop *reset* ha estat desactivat, el *buffer* comença a funcionar. Primerament, introdueix el bit zero "registre(longitud-2 downto 0)" al següent bit per deixar lliure un espai i poder introduir el nou bit que prové del senyal "in_sig" a "registre(0)". D'aquesta manera es van desplaçant els bits un a un des de l'entrada fins a la sortida del *buffer* obligant-los a entrar a l'*slave* SPI individualment.

Un cop el *buffer* ha fet la seva funció el senyal "out_sig" és el que porta el valor a l'*slave* i es connecta a la variable que pertoqui. Per produir aquesta connexió es creen uns ports entre els sincronismes i el propi *slave*. Els ports en qüestió s'insten de la següent manera:

```
-- instanciacio dels sincronitzadors pels senyals spi
--  l'objectiu es obtenir els mateixos senyals del bus spi
--  pero sincronitzats amb el clk del sistema (clk)

synch_SCLK : synchronizer -- sincronitzador spi_clk
   generic map (
      longitud => 2)
   port map (
      clk => clk,
      rst => rst,
      in_sig => spi_CLK,
      out_sig => sclk);

synch_MOSI : synchronizer -- sincronitzador spi_MOSI
   generic map (
      longitud => 2)
   port map (
      clk => clk,
      rst => rst,
      in_sig => spi_MOSI,
      out_sig => mosi);

synch_CS : synchronizer -- sincronitzador spi_CS
   generic map (
      longitud => 2)
   port map (
      clk => clk,
      rst => rst,
      in_sig => spi_CS,
      out_sig => cs);
```

Figura 5.11. Connexió dels ports entre sincronitzadors i *slave* SPI.

Cadascun d'aquests sincronitzadors s'utilitza per una de les variables d'entrada que proporciona el *master* de l'SPI. Com el procediment per processar la dada d'entrada és la mateixa en els tres casos es pot aprofitar la mateixa estructura genèrica del codi de sincronisme i posteriorment canviar només les connexions entre les variables.

Els senyals que es troben a la part esquerra de la fletxa són els senyals que estan incorporant-se des del *buffer* cap al *slave* i a la part dreta es troben els senyals on s'introduiran els valors un cop processats.

Com es pot comprovar, tant el senyal de rellotge com el de *reset* són comunes en els tres sincronismes. Això implica que el codi es mantindrà síncron en tot moment i quan es premi el botó de *reset* s'inicialitzarà tot el sistema alhora.

Les altres dues variables són les que varien segons el senyal que es vulgui introduir al *buffer.* Primerament s'instancia el senyal d'entrada, el qual és la que proporciona el propi *master* de l'SPI i, per tant, és el que es connecta amb "in_sig". D'altra banda, el senyal de sortida és el que ja s'ha passat pel *buffer* i es connecta amb "out_sig". Totes aquestes dades són senyals interns que només s'utilitzaran dins l'*slave* i serviran per iniciar els processos esmentats prèviament en aquest apartat.

A continuació es mostra el funcionament del sincronitzador del senyal de rellotge de l'SPI:

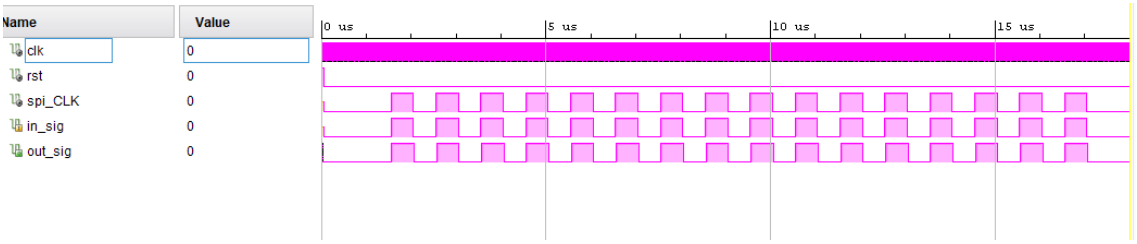

Figura 5.12. Simulació del sincronisme de la senyal de rellotge de l'SPI.

Tot i que semblen senyals idèntics, la variable de sortida "out_sig" està desplaçada amb un petit retard de 15 ns. En la imatge inferior es mostra un zoom on es veu la diferència.

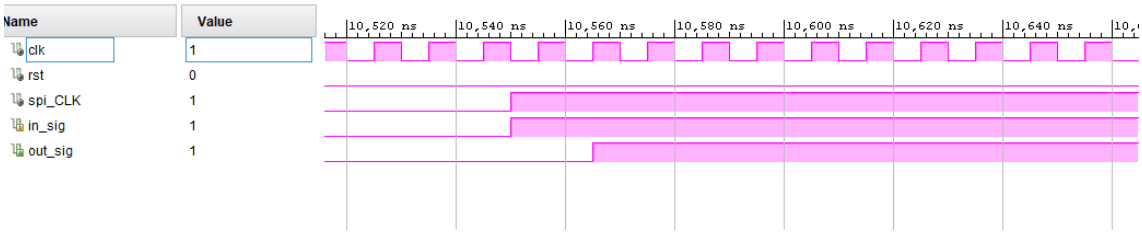

Figura 5.13. Zoom de la simulació del sincronisme.

5.3 Arxiu de restriccions

L'arxiu de restriccions o *Constraints* té la funció de connectar els senyals (tant d'entrada com de sortida) del codi amb la placa Nexys 4. És la manera que té el programa Vivado d'assegurar que les variables que es sol·liciten per enviar a la placa s'introdueixen en ports que existeixen en la placa i,

alhora, no estan protegits. Els ports protegits serien aquells imprescindibles pel correcte funcionament de la placa, per exemple els ports per on s'introdueix el voltatge per encendre la placa o el botó de *reset* general incorporat a la pròpia placa.

En aquest projecte s'utilitzen uns ports externs que serveixen per a introduir o extreure les variables que s'hagin processat dins la FPGA. Alhora també s'utilitzen els LEDs, els visualitzadors de set segments i els interruptors que incorpora la placa per comprovar que s'envien les dades correctament.

Primerament, el botó de *reset,* del qual ja s'ha esmentat el funcionament anteriorment, s'instancia de la següent manera en l'arxiu de restriccions:

```
76  ##Buttons
77  ##Bank = 15, Pin name = IO_L11N_T1_SRCC_15,              Sch name = BTNC
78  set_property PACKAGE_PIN E16 [get_ports rst]
79      set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

Figura 5.14. *Constraint* del botó de *reset.*

En segon lloc, els interruptors serviran per a què, un cop enviades les dades des de l'ordinador i enregistrades en els diferents registres, es pugui alternar quin registre es visualitzarà en els LEDs de la placa o en els visualitzadors. També s'utilitzarà un interruptor per canviar de visualització LED a 7 segments..

```
### Switches
##Bank = 34, Pin name = IO_L21P_T3_DQS_34,                Sch name = SW0
set_property PACKAGE_PIN U9 [get_ports {sw[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
##Bank = 34, Pin name = IO_25_34,                          Sch name = SW1
set_property PACKAGE_PIN U8 [get_ports {sw[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
##Bank = 34, Pin name = IO_L23P_T3_34,                     Sch name = SW2
set_property PACKAGE_PIN R7 [get_ports {sw[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
#Bank = 34, Pin name = IO_L14P_T2_SRCC_34,                Sch name = SW15
    set_property PACKAGE_PIN P4 [get_ports {sw[15]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]
```

Figura 5.15. *Constraints* dels interruptors.

En tercer lloc, els LEDs mostraran els bits que s'hagin guardat en el registre que es vulgui visualitzar amb els interruptors.

```
## LEDs
#Bank = 34, Pin name = IO_L24N_T3_34,                    Sch name = LED0
set_property PACKAGE_PIN T8 [get_ports {led[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
#Bank = 34, Pin name = IO_L21N_T3_DQS_34,                Sch name = LED1
set_property PACKAGE_PIN V9 [get_ports {led[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
#Bank = 34, Pin name = IO_L24P_T3_34,                    Sch name = LED2
set_property PACKAGE_PIN R8 [get_ports {led[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
#Bank = 34, Pin name = IO_L23N_T3_34,                    Sch name = LED3
set_property PACKAGE_PIN T6 [get_ports {led[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
#Bank = 34, Pin name = IO_L12P_T1_MRCC_34,               Sch name = LED4
set_property PACKAGE_PIN T5 [get_ports {led[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
#Bank = 34, Pin name = IO_L12N_T1_MRCC_34,               Sch name = LED5
set_property PACKAGE_PIN T4 [get_ports {led[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
#Bank = 34, Pin name = IO_L22P_T3_34,                    Sch name = LED6
set_property PACKAGE_PIN U7 [get_ports {led[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
```

Figura 5.16. *Constraints* dels LEDs.

Tot i que en la imatge només es mostrin sis restriccions el total de LEDs disponibles són 16. Quan es vulguin visualitzar els registres, com tenen longituds d'entre 11 o 12 bits, se'n desactivaran els sobrants per no mantenir-los a l'espera de dades durant tot el procés de comprovació.

Quan es vulgui comprovar l'enviament del missatge complert (bit R/W, registre i missatge) s'utilitzaran els 16 LEDs per mostrar que les dades que s'estan enviant són les que s'ha sol·licitat inicialment.

Finalment, els senyals que s'introdueixen per part del MCP2210 s'instancien en uns ports externs anomenats "*Pmod Header JC*".

```
##Pmod Header JC
#Bank = 35, Pin name = IO_L23P_T3_35,                    Sch name = JC1
set_property PACKAGE_PIN K2 [get_ports {spi_clk}]
    set_property IOSTANDARD LVCMOS33 [get_ports {spi_clk}]
#Bank = 35, Pin name = IO_L6P_T0_35,                     Sch name = JC2
set_property PACKAGE_PIN E7 [get_ports {spi_cs}]
    set_property IOSTANDARD LVCMOS33 [get_ports {spi_cs}]
#Bank = 35, Pin name = IO_L22P_T3_35,                    Sch name = JC3
set_property PACKAGE_PIN J3 [get_ports {spi_mosi}]
    set_property IOSTANDARD LVCMOS33 [get_ports {spi_mosi}]
#Bank = 35, Pin name = IO_L21P_T3_DQS_35,                Sch name = JC4
set_property PACKAGE_PIN J4 [get_ports {spi_miso}]
    set_property IOSTANDARD LVCMOS33 [get_ports {spi_miso}]
```

Figura 5.17. *Constraints* dels senyals del MCP2210.

6 Interfície gràfica

La interfície gràfica és l'únic apartat de tot el programa que ha de mantenir un contacte directe amb l'usuari. Aquesta ha de suplir totes les necessitats de la persona que estigui controlant l'ordinador i per tant, s'ha d'aconseguir una fàcil lectura dels passos a seguir i que aquests siguin simples i efectius a l'hora de resoldre'ls.

Primer de tot, s'ha de permetre omplir totes les dades requerides per l'usuari de tal manera que quan s'enviïn les dades no quedi cap espai en blanc que pugui provocar un error en l'enviament d'aquestes.

El programa amb el que es codifica tota aquesta estructura s'anomena Visual Studio de la companyia Microsoft. Com és un programa de desenvolupament via objectes permet al programador dissenyar l'estructura visual i al mateix temps, poder indicar quines funcions es duen a terme quan s'interactua amb els diferents objectes.

En aquest apartat s'explicaran tant les funcions i les classes utilitzades per l'enviament de dades com el disseny i els diferents passos que s'han de dur a terme en el disseny visual. Com el codi supera les 1500 línies de programació no s'inclouran totes i cadascuna de les funcions en l'explicació però si que s'esmentaran en els resums de cada apartat.

En l'annex es podran trobar tots els codis utilitzats.

6.1 Disseny visual

El disseny visual és el grup de finestres amb els controls que l'usuari té a la vista com ja poden ser: botons, *textboxes*, *radiobuttons*...

La finestra principal s'anomena *Form* i és la que es crida quan s'inicia el programa. És el llenç on s'hi introduiran tots els controls. Es poden crear més d'un *form* en un mateix projecte però en aquest treball s'ha decidit aprofitar l'eina *Tabs* que ofereix el programa. *Tab* permet crear diferents finestres independents les unes de les altres i dóna la possibilitat de nombrar-les per tal de poder diferenciar-les correctament.

Les dues *tabs* que es fan servir són:

- **Enviament de dades:** Aquí s'introdueixen i visualitzen les sol·licituds de l'usuari: El missatge a enviar, la hora de l'enviament, els registres etc.
- **Configuració:** En aquesta finestra s'escullen els pins que es requereixin per l'enviament de dades. També es poden modificar les configuracions del protocol SPI com ara per exemple la velocitat de l'enviament, la quantitat de dades enviades o si es requereix algun retard en la captació d'aquestes.

6.1.1 Enviament de dades

L'estructura amb la que s'ha dissenyat aquesta pestanya és una que es pugui llegir de esquerra a dreta i permeti introduir totes les peticions requerides pel correcte funcionament. A continuació es mostra un diagrama amb les diferents opcions que es proposen a l'usuari:

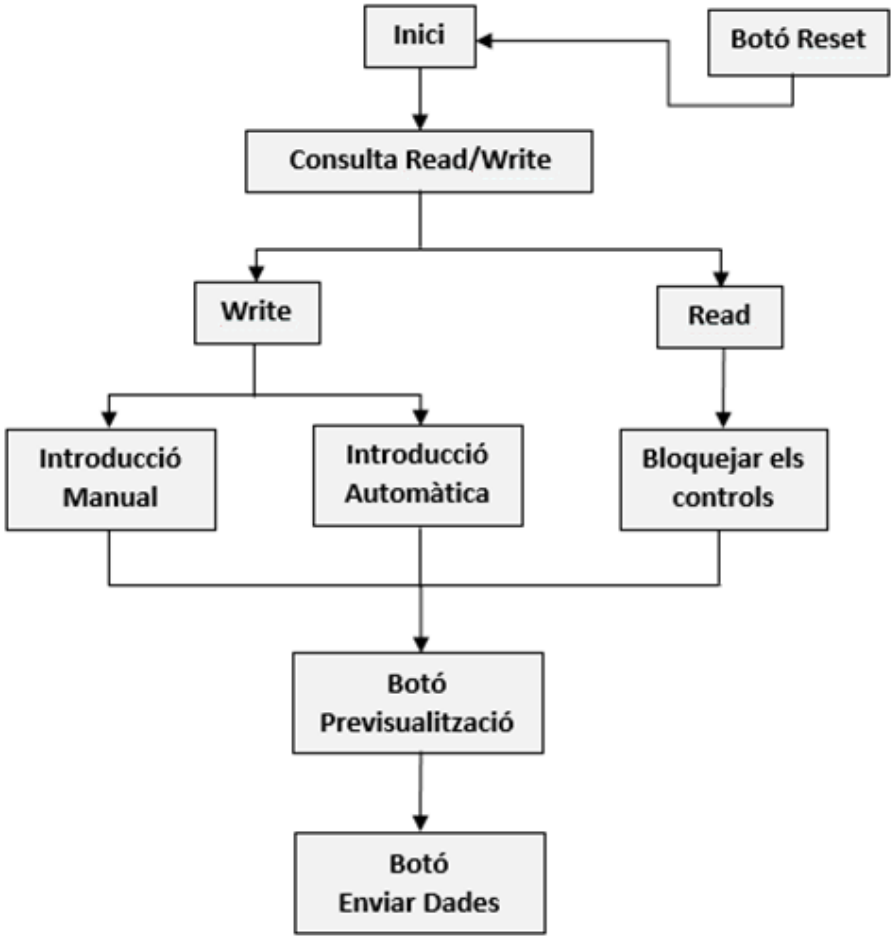

Figura 6.1. Diagrama de la pestanya "Enviament de dades".

Primer de tot, es sol·licita que s'esculli entre enviar o rebre dades de part de l'SPI. En el cas de que es demani una lectura dels registres, es bloquejaran tots els controls per evitar l'enviament i la recepció de dades simultàniament. Només es deixaran oberts els botons de previsualització i el d'enviament de dades.

Si d'altra banda, la intenció de l'usuari és enviar noves dades, escollirà *write* i se li permetrà introduir el missatge manualment o fent ús d'unes plantilles ja codificades prèviament.

L'objectiu final és introduir les dades a un codi on es sol·liciten un conjunt de bits de 35 bits de longitud, per tant, tot missatge que no compleixi aquests requisit serà denegat en la fase de previsualització.

Si s'escull la introducció de dades automàtica, només caldrà seleccionar la plantilla que es desitgi i si es vol enviar amb o sense soroll en la imatge. El soroll és una manera de diferenciar les imatges completes de les incompletes (varien d'entre 3 o 4 bits en total).

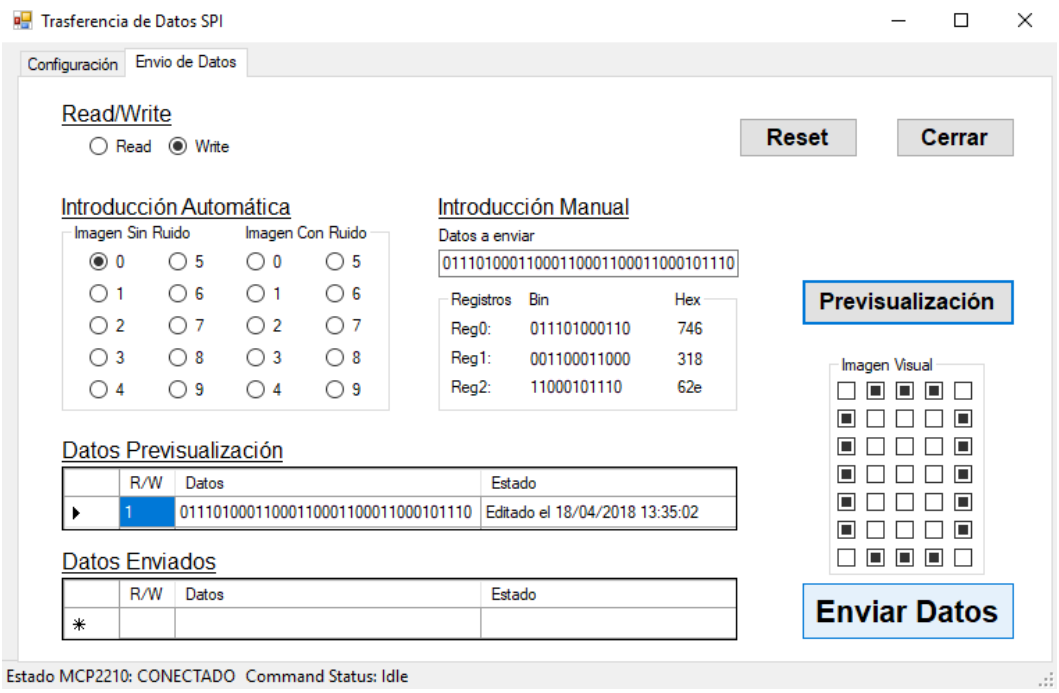

Figura 6.2. Enviament de dades. Previsualització correcte.

La imatge mostra com quedaria la finestra un cop el botó Previsualització ha estat premut i no ha sorgit cap error que impedeixi la continuació del procés. Encara que es seleccioni la introducció automàtica de dades, es pot observar que el missatge de 35 bits també es mostra en la introducció manual. Això és a causa de que es més senzill modificar un parell de bits d'una plantilla que no pas introduir els 35 bits un a un.

Seguidament es mostren altres exemple de com es mostrarien les imatges si es modifiquessin els bits.

Figura 6.3. Enviament de dades. Exemples d'imatges del nombre 0 amb soroll.

Un cop s'ha confirmat la previsualització, s'actualitzarà la barra d'informació de "Dades Previsualitzades" amb les dades introduides. S'inclourà la direcció (*Read/Write*), el missatge que es vol enviar i l'hora de la previsualització. Si tot és correcte aleshores es pot prémer el botó d'enviar dades el qual actualitzarà la barra que correspongui a la sol·licitud.

Finalment, s'ha incorporat un botó de *Reset* per si es vol tornar a començar el procés el qual borrarà tota la informació dels controls de la finestra.

6.1.2 Configuració

En aquesta finestra es troben tots els controls que modifiquen la configuració necessària per a què el protocol d'enviament SPI funcioni adequadament.

Per una part, es pot escollir quin *slave* estarà actiu a l'hora de carregar les dades. Pot haver-hi més d'un actiu alhora per si es requerissin diversos dispositius connectats al *master,* però en aquest projecte ho limitarem a un dispositiu *slave* que rebrà totes les connexions del *master*.

Alhora, s'hi pot configurar les diverses característiques pròpies del protocol SPI com ja s'ha esmentat en apartats anteriors.

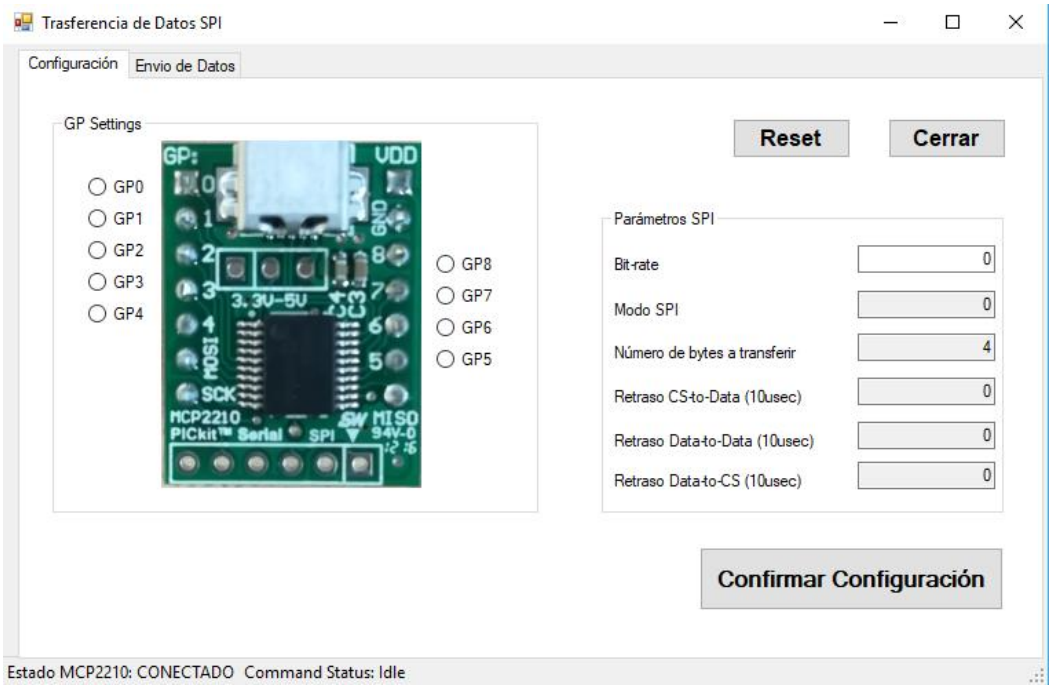

Figura 6.4. Configuració Interfície gràfica

6.2 Classes

Quan els codis a programar comencen a expandir-se és molt recomanable utilitzar algun tipus d'organització interna per trobar cada apartat el més ràpid possible. En el Visual Studio, la millor manera per estructurar el codi és crear diferents classes que reparteixin les diferents funcions de tot el conjunt.

En aquest projecte s'ha utilitzat com a base inicial el codi de comunicació SPI que proporciona la companyia Microchip i incorpora totes les funcions necessàries per a una correcte connexió amb la placa MCP2210. Així doncs, les funcions que corresponen a la verificació dels paràmetres SPI i l'execució de la comunicació s'han extret de dit codi.

D'altra banda, la classe "Controls" s'ha basat a partir del codi de la Núria Torres, el qual realitzava unes tasques similars a les que aquest projecte requeria. Tot i haver basat aquesta classe segons el codi de la Núria, s'han fet varies modificacions per que s'adeqüi a les necessitats del projecte que s'ha dut a terme.

En el projecte s'hi troben varies classes que agrupen la totalitat del codi:

- **Form Terminal:** És la classe principal. Incorpora totes les interaccions amb els controls del disseny visual (clics de botons, selecció de *radiobuttons*...). També s'hi sol·liciten les actualitzacions de les barres d'informació un cop realitzades totes les comprovacions corresponents amb el botó Previsualització.
- **Classe MCP2210:** Aquesta classe permetrà al programa controlar tota la configuració del MCP2210: estat de la connexió, transferència i obtenció de la informació sobre l'SPI i detecció de qualssevol interrupció durant el procés de configuració.
- **Classe Data Element:** En aquest apartat s'hi troben totes les variables que s'hauran d'actualitzar abans d'introduir-les a les barres d'estat de la finestra principal.
- **Classe BytesStructure:** Aquí s'hi verifica la estructura de les barres de dades. Segons si es sol·licita *Read/Write* o la comprovació de que les dades que es pretenen enviar tenen els paràmetres correctes.
- **Classe Controls:** Classe que controla el contingut de les barres d'informació i alhora avisa si hi ha algun problema amb l'actualització d'aquestes. Aquí s'hi troben totes les verificacions i els avisos pertinents en cas de que no resultin totes correctes.

6.2.1 Form Terminal

Aquesta classe conté totes les funcions relacionades amb la interfície gràfica que s'ha dissenyat. Té una sèrie d'objectes generals que seran essencials per a què la classe i les funcions que s'utilitzin en el seu interior puguin ser cridades.

6.2.1.1 Objectes Generals

- **MCP2210:** Crea una instància per a la classe MCP2210 i així es puguin cridar les funcions que hi ha dins aquesta classe en referència a la transferència de dades SPI.
- **BytesStructure:** Crea una instància per a la classe BytesStructure que contindrà tota la informació relacionada amb l'estructura de barres d'estat.
- **FormControls:** Crea una instància per a la classe Controls on s'hi inclouran totes les dades que s'introduiran a les barres d'estat posteriorment.

6.2.1.2 Funcions de la classe

- **Form Load:** Funció per a obrir la finestra principal un cop es carrega el codi. Inicialitza la interfície gràfica, actualitza la configuració de quins *chip-select* es requereixen i inicialitza el *timer* que anirà visualitzant si el MCP2210 està o no connectat a l'ordinador.
- **Timer Tick:** Cada milisegon comprova si el hardware MCP2210 està connectat i ho escriu a la part inferior del disseny, a la barra d'informació general.
- **Update Bytes For Transfer:** Funció que rep les dades introduïdes per a l'enviament de dades i les incorpora a la barra de dades de previsualització. En el cas de que falti alguna dada, saltarà un error informant de que no s'ha pogut completar la configuració prèvia a l'enviament.
- **Update Grid After Transfer:** Similar a l'anterior funció però aquesta s'actualitza un cop s'han enviat les dades. Aquesta funció actualitza la barra de dades enviades o rebudes en funció de la sol·licitud de l'usuari. Si no estigués connectat el MCP2210, sortiria un error avisant de que es requereix un MCP2210 connectat per procedir a l'enviament o recepció de dades.
- **Reset All Controls:** Funció de Reset on es netegen tots els controls que hi ha a la finestra principal. Borra totes les seleccions i dades escrites de tal manera que es pugui reiniciar el procés de nou sense haver de fer cap modificació manual.

```
66  Private Sub ResetAllControls(ByVal container As Control)
67      For Each ctrl As Control In container.Controls
68          If TypeOf ctrl Is RadioButton Then
69              DirectCast(ctrl, RadioButton).Checked = False
70          ElseIf TypeOf ctrl Is TextBox Then
71              DirectCast(ctrl, TextBox).Clear()
72          ElseIf TypeOf ctrl Is CheckBox Then
73              DirectCast(ctrl, CheckBox).Checked = False
74          ElseIf TypeOf ctrl Is DataGridView Then
75              DirectCast(ctrl, DataGridView).Rows.Clear()
76          End If
77          If ctrl.Controls.Count > 0 Then
78              ResetAllControls(ctrl)
79          End If
80      Next
81  End Sub
82
83  Private Sub Button_Reset_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
84      ResetAllControls(TabEnvioDatos)
85      Button_TxferSPIData.Enabled = False
86  End Sub
```

Figura 6.5. Form Terminal. Funció de Reset.

- **Button Previsualizacion:** Un cop es premi el botó de previsualització, s'actualitzaran tant la barra d'estat corresponent com la imatge visual que es pretén enviar.
- **Button Enviar/Rebre Dades:** Última funció de la classe principal. Recull la configuració de l'SPI, envia les dades cap al MCP2210 i finalment, actualitza la barra d'estats que pertoqui segons l'acció que s'hagi completat.

6.2.2 MCP2210

Com ja s'ha esmentat anteriorment, aquesta classe defineix tots els paràmetres que configuren el MCP2210 i l'execució de la comunicació.

6.2.2.1 Objectes Generals

- **VID i PID:** Constants necessàries per a la comunicació SPI.
- **NUM_GP:** Nombre de GPIO disponibles. En aquest cas el MCP2210 en té un total de 9.
- **USB SPI:** Objecte amb la funció de connectar el MCP2210.
- **MCP2210 Connection:** Variable booleana que representa la connexió correcta del MCP2210.
- **TxData i RxData:** Arrays de bytes que s'envien o es reben segons la petició de l'usuari.

6.2.2.2 Funcions de la classe

- **Update SPI Parameters:** Actualitza les variables de configuració del MCP2210 a partir de les dades introduïdes a la finestra "Configuració".
- **Check USB SPI Connection:** Aquesta funció és la que revisa si el MCP2210 està connectat al USB de l'ordinador cada milisegon del rellotge abans esmentat.
- **Transfer SPI Data:** Aquí és on el missatge pren forma i es registra en diferents variables per després concatenar-les i enviar-les via SPI. Només es pot entrar quan la connexió amb el MCP2210 ha estat validada. En aquesta funció es llegeix el contingut del missatge que es pretén enviar i s'introdueix a la variable TxData o RxData segons la petició. Com el missatge són 35 bits concatenats, es repetirà la introducció de bits a TxData tres vegades per tal d'aconseguir la longitud necessària. D'altra banda, també s'actualitzarà l'estat amb l'hora que s'introduirà a la barra d'estats.
- **Communication Execution:** En aquesta funció s'atura el rellotge durant uns instants per a fer l'enviament de dades via SPI amb totes les configuracions prèviament seleccionades i revisades. Si sorgís algun error en l'enviament, aquesta funció crearia una finestra d'error on avisaria de quina variable s'ha deixat en blanc o fora de rang que ha provocat la fallida del sistema.

6.2.3 Data Element

Aquesta classe s'utilitza per agrupar tota la informació relacionada amb l'element a enviar, incloent el bit de escriptura o lectura, el missatge a enviar en 1 i 0, l'estat de l'enviament i l'hora d'aquest. Per guardar tota aquesta informació es creen diferents variables les qual s'esmenten a continuació:

- **ReadWrite:** Bit de lectura o escriptura segons si s'introdueix un 0 o un 1.
- **DataByte:** *String* on el missatge es guarda previ a l'enviament.
- **Status:** *String* on s'hi inclou tant l'hora de l'enviament com si s'ha llegit o escrit.

En aquesta classe no hi ha cap funció ja que no s'utilitza pel comportament de cap element, senzillament serveix per estructurar les dades ja seleccionades.

6.2.4 BytesStructure

Aquí es troben les agrupacions d'elements creades dins la classe Data Element. El paràmetre que s'utilitza en aquesta classe és:

- **List Data Elements:** Llista que conté els objectes desitjats creats per la classe Data Element.

6.2.5 Controls

Per últim, aquesta classe conté tota la generació i neteja del contingut dels panells de controls, les actualitzacions dels seus valors, de les barres d'estat de previsualització i enviament/recepció.

6.2.5.1 Funcions de la classe

- **Update Data:** Actualitza el contingut de les barres d'estat. Revisa si les dades introduïdes són correctes, i si ho són, les traspassa a la barra d'informació de previsualització.
- **Update After Transfer:** Actualitza els valors que s'introduiran a les barres d'estat d'enviament o recepció.

7 Implementació i test

Com que el projecte es basa en el desenvolupament de diferents mòduls connectats entre ells, primer de tot es simularan cadascun dels mòduls per separat i a continuació es simularan en conjunt.

Per a testejar el comportament de la placa MCP2210 s'utilitzarà un mòdul de test que proporciona Microchip per observar les variables que envia i rep cada pin de la placa.

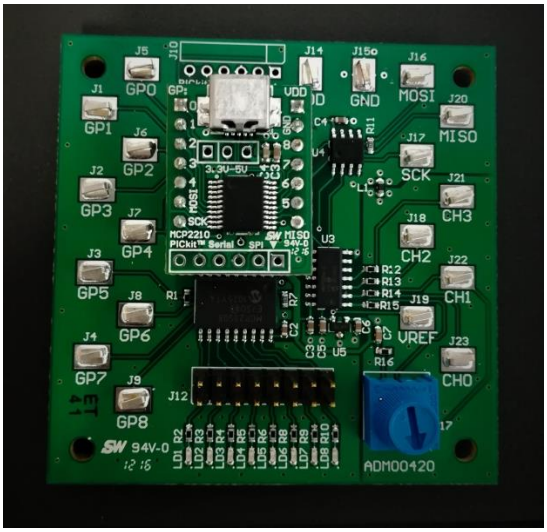

Figura 7.1. Placa MCP2210 connectada a la seva placa de proves.

Finalment, el codi en VHDL es simularà en el propi programa Vivado i seguidament es traslladarà a una variable de sortida com podrien ser els LEDs de la FPGA.

7.1 Test del MCP2210

Com ja s'ha esmentat, es permetrà visualitzar les diferents variables del MCP2210 amb una placa externa de test. Aquesta placa permetrà testejar les diferents funcions i permet observar com es configuren els pin de *chip-select*, els senyals d'entrada i, alhora, les variables de sortida amb el resultat que es desitgi.

El primer test que es realitzarà amb la placa serà amb un programa que proporciona Microchip on es permet modificar tots els paràmetres per l'enviament de dades via SPI.

El programa permet enviar una sèrie de bytes que s'introduiran via MOSI i a l'altra part s'hi mostraran les respostes que emet el MISO. D'altra banda, s'observaran com han quedat configurats els GPs i la seva configuració.

Pel test, s'enviaran dos bytes per comprovar que la comunicació és correcte i les dades es transmeten correctament. Així doncs, es configurarà l'enviament per a 2 bytes mentre que la resta de paràmetres es mantindran amb els seus valors per defecte.

En aquest cas, el *chip-select* que s'activarà serà el GP0, mantenint la resta inoperatius durant el test.

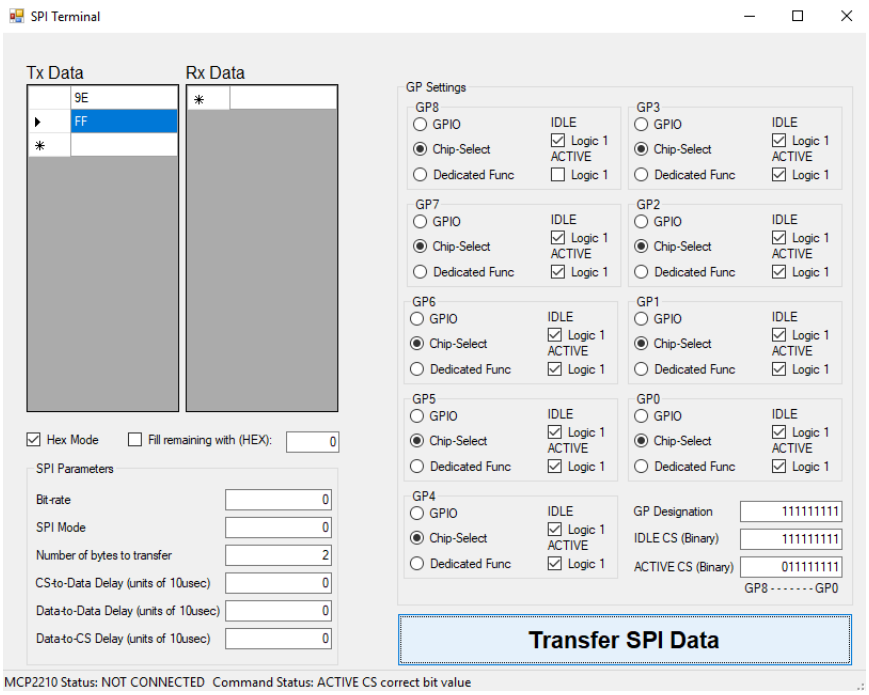

Figura 7.2. Test enviament dades.

Els senyals que es poden visualitzar en l'oscil·loscopi són tant el SCLK com el MOSI on s'hi mostren les dades enviades cap a la placa. A continuació es mostra una imatge amb el resultat de l'enviament de les dades introduïdes anteriorment:

- Missatge: 9E FF = 1001111011111111

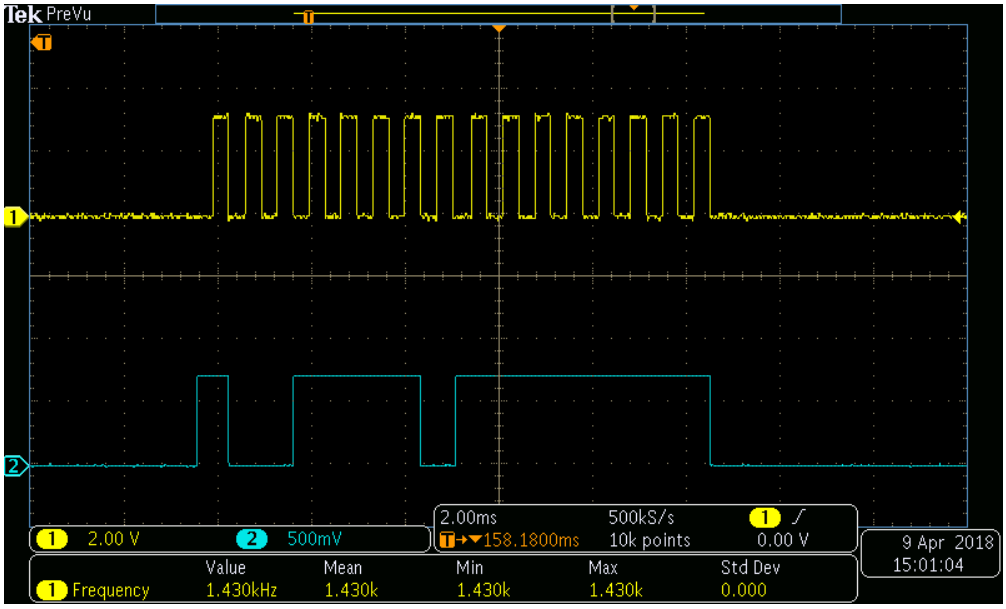

Figura 7.3. Senyals SCLK i MOSI del MCP2210 quan s'envien 9E i FF.

Amb aquesta demostració podem confirmar que la placa funciona correctament i permet l'enviament de les dades que se li sol·liciten.

7.2 Test del codi VHDL

Pel test del codi VHDL s'utilitzarà la placa FPGA Nexys4 on hi ha incorporats una sèrie de LEDs i interruptors que permetran visualitzar les dades que s'introdueixin. Posteriorment, s'inclourà la visualització en els 7 segments que incorpora la Nexys4.

Un cop s'hagin realitzat les simulacions i comprovat que les dades es reben i s'emmagatzemen correctament, es traslladarà el codi a la següent fase. Aquesta fase és l'anomenada implementació on s'introdueix el programa a la FPGA, la qual queda preparada per a un nou enviament de dades per part del MCP2210.

Abans d'enviar les dades des del MCP2210, primer es substituiran els senyals d'entrada per variables fixes per assegurar que el procés d'emmagatzematge és el correcte i les dades es guarden als registres que els hi pertoquen. D'aquesta manera es podrà comprovar que la visualització funciona i apareixen els valors que se'ls ha proporcionat.

D'altra banda, per comprovar que els senyals de sortida realitzen les seves tasques correctament, se'ls substitueix per la tira de LEDs que hi ha a la placa de tal manera que quan s'incorporin noves dades, es visualitzin en aquests.

Per portar-ho un pas més enllà i que la visualització sigui més agradable per a l'usuari, també es codificarà la visualització pels 7 segments. Per corroborar que els resultats que es mostren a la placa són els enviats, a la interfície gràfica es podrà visualitzar una conversió a hexadecimal del missatge que s'ha enviat.

Seguidament s'ensenyen els diferents resultats enregistrats i visualitzats en els LEDs i els 7 segments de la FPGA:

- Missatge: "11111100001111000001000011000101110"
- Missatge en hexadecimal: "FC3C1062E"
    - Reg0 = "FC3" – "111111000011"
    - Reg1 = "C10" – "110000010000"
    - Reg2 = "62E" – "11000101110"

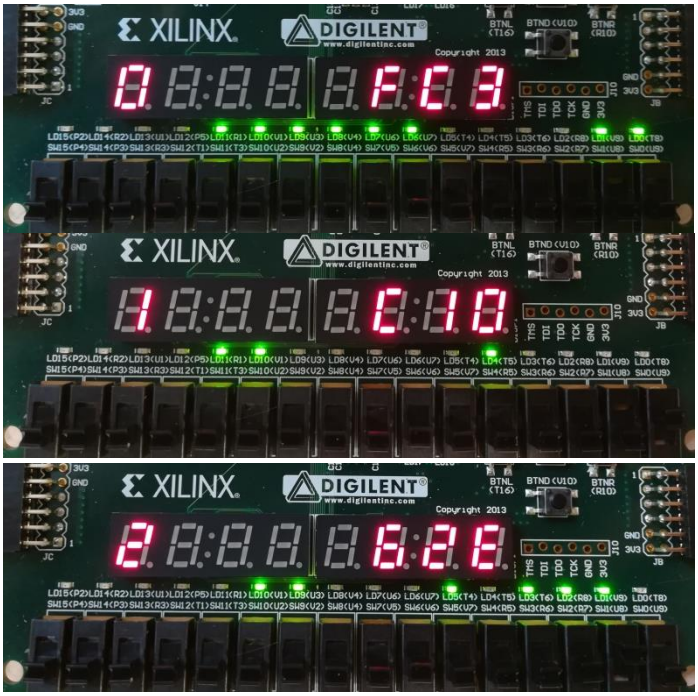

Figura 7.5. Visualització del missatge via LEDs i 7 segments.

7.3 Test de la interfície gràfica

Per testejar la interfície gràfica s'utilitzarà l'oscil·loscopi on s'hi podran veure no només els senyals de rellotge i de sortida MOSI sinó que també es podrà observar el senyal de *chip-select* que permet tot el procés d'enviament.

Primer de tot, es configuraran tots els paràmetres per la connexió SPI i es seleccionarà el *chip-select* que es vulgui activar per transmetre les dades. Un cop s'hagi configurat la connexió SPI, s'escriuran manualment les dades en el *textbox* corresponent i s'enviaran cap al MOSI.

- Missatge: 1101100000101110

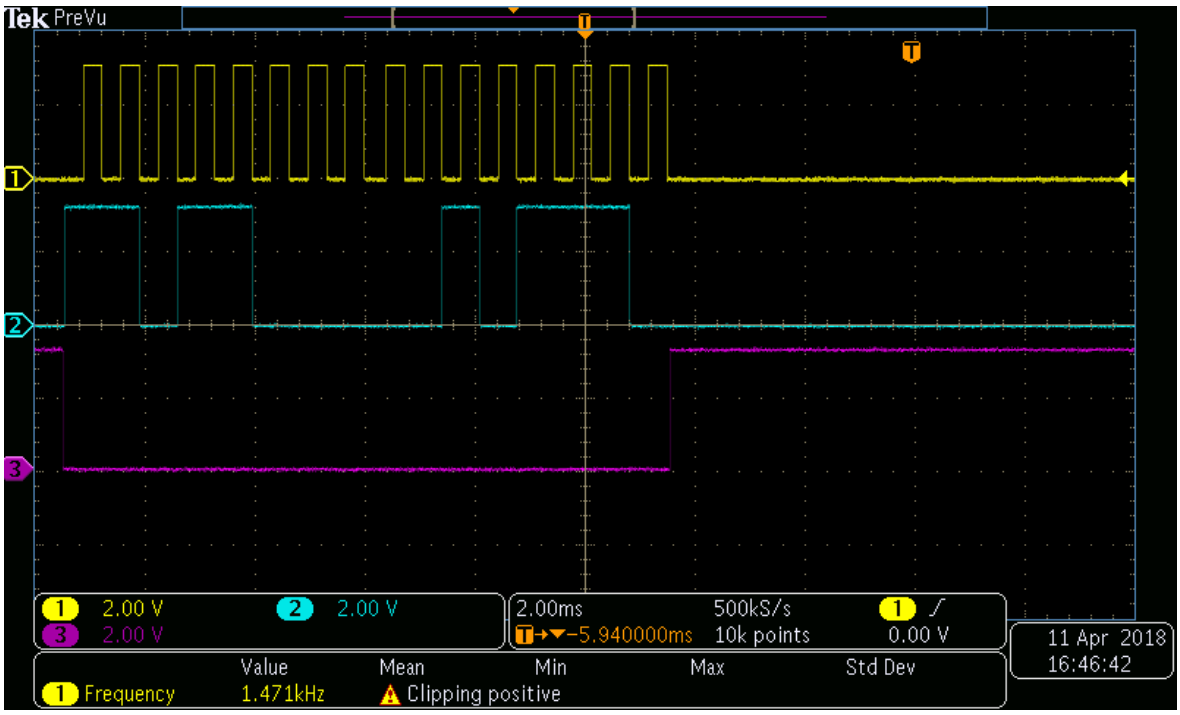

Figura 7.6. Senyals de rellotge,MOSI i *chip-select* visualitzades via oscil·loscopi

Conclusions

Aquest projecte m'ha permès desenvolupar dos codis completament diferents i, inicialment inconnexes, per posteriorment crear una connexió entre elles que ha permès l'enviament de dades d'un a l'altre.

Per la part que comporta la interfície gràfica, s'ha pogut resoldre tant el disseny com el desenvolupament gràcies en part a un programa de Microchip. Aquest programa porta incorporades les funcions bàsiques de l'enviament de dades per comunicar-se amb el MCP2210 .Amb aquest apartat del projecte s'ha hagut d'anar amb compte de no capficar-se amb detalls visuals i enfocar-se amb el funcionament i l'enviament de les dades cap al MCP2210.

El mòdul VHDL inicialment va comportar varies dificultats derivades del protocol de comunicació SPI, però finalment es va aconseguir la connexió desitjada entre *master-slave*. Quan van estar enllestides tant la part de comunicació com la part de l'emmagatzematge, l'estructura del projecte ja va a consolidar-se, i a partir d'aquell punt ja va començar a esclarir-se el camí. D'altra banda, també he aconseguit més soltesa amb la programació VHDL, millorant les meves habilitats i assentant millor les bases apreses en diferents assignatures de la carrera.

Tot i que a classe es va utilitzar el programa ISE per implementar en les FPGAs els codis realitzats, en aquest projecte s'ha utilitzat l'última versió del programa VIvado. Aquest fet va comportar un estudi previ per comprendre les diferències que existien entre ambdós programes. Un cop feta la revisió de l'estat de l'art i solucionades les discrepàncies entre els programes es va començar el disseny, desenvolupament i test. Aquest aprofundiment amb la codificació en VHDL i la seva implementació m'ha permès entendre millor com funciona la introducció dels codis en placa FPGA i quins han de ser els passos a seguir per tal d'evitar errors en aquest procés.

És per això, que aquest projecte m'ha donat l'oportunitat de poder desenvolupar una aplicació d'inici a fi, la qual m'ha proporcionat molta experiència amb el disseny d'interfícies gràfiques.

Per resumir finalment, aquest projecte m'ha permès investigar els dos camps (disseny d'interfícies gràfiques i implementació en plaques FPGA) els quals havia vist molt breument al llarg de la carrera però sempre m'havien semblat dels més interessants de tot el sector de l'enginyeria electrònica.

Objectius resolts

Durant aquest projecte, s'han completat diversos objectius dels que es van proposar inicialment, en la preparació prèvia del treball. D'altres no s'han pogut resoldre per diferents motius com són la manca de recursos o la falta de temps per les dates límit per l'entrega del projecte.

Els objectius inicials eren:

- Recopilar informació referent als diferents tipus de estàndards de comunicació i decidir el més adient per les necessitats que es proposen pel projecte.
- Dissenyar el codi VHDL que permeti connectar el PC amb la FPGA.
- Adaptar el codi VHDL per permetre modificar imatges de dimensions 5x7.
- Programar una interfície gràfica eficient per configurar la connexió prèviament esmentada.
- Augmentar la capacitat de modificació del codi per imatges de 28x28.

Recopilar informació

Aquest objectiu era un requeriment essencial per poder dur a terme el treball d'una forma estructurada i amb una base suficientment sòlida com per poder treure endavant qualssevol problema que pogués comportar la connexió entre la interfície gràfica, el MCP2210 i el codi VHDL.

Dissenyar el codi VHDL

El disseny ha funcionat correctament un cop les dificultats que proporcionava el protocol de connexió ha estat resoltes. Inicialment es van fer una sèrie de diagrames per comprovar que no es deixava cap funció fora del codi i quedava tot ben connectat.

Finalment aquest programa ha estat capaç de:

- Activar-se només al rebre dades, estalviant energia al evitar la connexió contínua.
- Rebre la informació que enviava el *master* MCP2210.
- Emmagatzemar dita informació en diferents registres interns al codi.
- Concatenar el resultat introduït en els registres.
- Enviar el resultat final a l'exterior per visualitzar-lo en la FPGA.

Adaptar el codi per imatges 5x7

Aquest objectiu inicialment tenia la intenció de modificar la recepció de dades per aconseguir emmagatzemar 35 bits i llegir-los com un nombre predeterminat. Aquesta idea es va veure alterada i es va decidir transportar-la a la interfície gràfica, la qual és molt més mal·leable i permet enviar les dades que es sol·liciten i que alhora pot mostrar visualment la imatge prèviament a l'enviament.

Tot i així, s'ha portat a terme sense que sorgís cap conflicte amb l'enviament de dades ja establert ni amb l'emmagatzematge d'aquestes mateixes dades.

Programar una interfície gràfica

En aquest projecte s'ha programat una interfície senzilla i resolutiva la qual permet enviar o rebre dades segons l'usuari sol·liciti. Està programada per evitar qualssevol error humà i té múltiples verificacions prèvies a l'enviament de les dades cap a l'*slave* SPI, assegurant d'aquesta manera que el missatge que es vol transmetre i enregistrar és exactament el que l'usuari està enviant.

Les proteccions que s'han proporcionat en aquesta interfície gràfica són:

- Inhabilitació de totes les parts fins que no es seleccioni enviament o recepció de dades.
- Habilitació de les diferents parts necessàries segons la selecció prèvia, evitant d'aquesta manera que quan es sol·liciti recepció de dades, l'usuari pugui introduir dades en les diferents barres d'escriptura.
- Botó de *Reset* que permet reiniciar tot error que l'usuari vulgui corregir abans d'enviar les dades cap a l'*slave*.

Connectar el conjunt a una xarxa neuronal ja dissenyada.

La intenció inicialment era que, un cop enllestida la connexió entre la interfície i la FPGA, es pogués introduir el missatge enviat a una xarxa neuronal que rebés i processes les dades. Aquest objectiu, finalment no s'ha pogut assolir perquè en el moment d'incorporar les dades resultants de la connexió SPI la xarxa neuronal no les rebia i retornava una sèrie de variables amb valor '0'.

Així doncs, tot i que les dues parts funcionen correctament quan se les fa treballar per separat, en el moment d'ajuntar-les es creen certes connexions errònies que impedeixen el traspàs de les dades que es sol·liciten.

Augmentar la capacitat fins a 28x28

Aquest objectiu volia proporcionar una expansió del projecte de la xarxa neuronal, ja que no només s'haurien d'enviar i enregistrar els 784 bits sinó que també s'hauria de modificar el codi de la xarxa neuronal per tal de que pogués rebre aquesta quantitat d'informació alhora i la pogués processar adequadament.

Pressupost i/o Anàlisi Econòmica

Els projectes d'enginyeria acostumen a tenir costos elevats principalment per les llicències que s'ahn de comprar per realitzar el projecte i les hores que s'han de posar per realitzar-lo.

És per això que, el pressupost es dividirà en els següents apartats:

- Software.
- Hardware.
- Hores d'enginyeria.

Llicències

En aquest projecte s'ha utilitzat diverses aplicacions, els quals requereixen d'una llicència (disponible a la UPC) per poder ser utilitzats.

Taula P1. Costos de les llicencies utilitzades.

	Unitats	Preu/Unitat	Total
Visual Studio Professional 2017	1	641,00 €	641,00 €
Xilinx HL Design Edition	1	2425,59 €	2425,59 €
Microsoft Office 2010	1	99,00 €	99,00 €
TOTAL			**3165, 59 €**

Hardware

Pel desenvolupament d'aquest projecte s'han fet servir els següents materials físics:

- FPGA Nexys 4.
- Placa MCP2210 i la seva placa de proves.
- Ordinador Asus GL702.

Els costos d'aquests elements es desglossen a continuació:

Taula P2. Costos dels materials utilitzats.

	Unitats	Preu/Unitat	Total
FPGA Nexys 4	1	259,16 €	259,16 €
Placa MCP2210	1	24,49 €	24,49 €
Ordinador	1/8	1000 €	250 €
TOTAL			**533,65 €**

El preu de l'ordinador s'ha calculat segons la vida útil que s'acostuma a atribuir a aquests aparells. Si la vida útil d'un ordinador l'associem a quatre anys i aquest projecte s'ha dut a terme en un termini de mig any, el cost que consta a la taula és l'equivalent a un vuitè de la vida útil.

Hores d'enginyeria

Pel disseny d'ambdós codis realitzats en aquest projecte, s'han invertit certes hores d'enginyeria per portar-los a terme. Aquest cost es desglossa en el diferents apartats que s'han realitzat al llarg del projecte.

Taula P3. Costos dels temps d'enginyeria.

	Hores	Preu/Hora	Total
Estudi SPI i MCP2210	80	7,00 €	560,00 €
Desenvolupament de la interfície gràfica	200	12,00 €	2.400,00 €
Desenvolupament del codi VHDL	250	12,00 €	3.000,00 €
Test	150	7,00 €	1.050,00 €
TOTAL	**680**		**7.010,00 €**

Pressupost total

Un cop s'han desglossat tots els apartats del pressupost, a continuació es mostra una suma del total que comportaria econòmicament realitzar aquest projecte.

Taula P4. Pressupost total.

	Preu Total
Llicencies	3.165,59 €
Hardware	533,65 €
Hores d'enginyeria	7.010,00 €
TOTAL	**10.709,24 €**

El preu s'eleva en el moment que es requereixen *softwares* de pagament i amb llicències. Totes les llicències es poden obtenir contactant amb les empreses que tenen els drets dels programes. Aquestes ofereixen descomptes pels a alumnes que estiguin cursant enginyeries i tinguin intenció d'utilitzar el programa per un període de temps inferior a un any.

El cost de les hores d'enginyeria s'ha establert segons els preus actuals d'enginyers joves recent graduats de la universitat.

Les hores dedicades a cada apartat del projecte s'han estimat partint de que el projecte ja estava iniciat, però en l'apartat de test s'han afegit hores de més a causa de les diverses dificultats que s'han trobat al llarg del desenvolupament.

Bibliografia

[1] Quanser Inc. 2017. "QUARC Communications Protocols .SPI Protocol"
http://quanser-update.azurewebsites.net/quarc/documentation/net/html/spi_protocol.html

[2] Microchip Technology. 2018. "MCP2210 Datasheet".
https://www.microchip.com/wwwproducts/en/MCP2210

[3] López Pérez, Eric. "Ingeniería en Microcontroladores. Protocolo SPI".
http://www.i-micro.com/pdf/articulos/spi.pdf

[4] Srinivas, Ananthula. Kiran, M., Kishore, Jugal. 2014. "Design and Verification of Serial Peripheral Interface".
https://www.ijedr.org/papers/IJEDR1303026.pdf

[5] Anònim. 2017. "Programación en VHDL. Capítulo 3: Arquitectura".
https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_VHDL/Arquitectura

[6] Tylee, Lou. 1998. "Learn Visual Basic 6.0".
https://www.uop.edu.jo/download/research/members/vb6_1__1_.0%20-%20visual%20basic%20-%20learn%20visual%20basic%206.0%20(nice%20manual).pdf

[7] Xilinx. 2013. "Nexs4 FPGA Board Reference Manual".
https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2013x/Nexys4/Supporting%20Material/Nexys4_RM_VB1_Final_3.pdf

[8] Anònim. 2012. "Button to uncheck all RadioButtons and clear all textfields in form"
http://www.vbforums.com/showthread.php?689897

[9] Xilinx. 2012. "Vivado Design Suite User Guide. Design Flows Overview."
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_4/ug892-vivado-design-flows-overview.pdf

[10] Voon Kiong, Liew. 2008. "Visual Basic 6 Tutorial."
http://www.vbtutor.net/vbtutor.html

[11] Van Loi Le. 2017. "VHDL code for Seven-Segment Display on Basys 3 FPGA".
http://www.fpga4student.com/2017/09/vhdl-code-for-seven-segment-display.html

Annex A

A1. Manual d'usuari

Aquest programa permet realitzar totes les accions esmentades a continuació:

- Configurar el MCP2210.
- Configurar el missatge a enviar.
- Enviar les dades cap a la placa FPGA.

- Passos previs

Previ a l'enviament de dades s'hauran de fer una sèrie de passos per assegurar que l'enviament es pot realitzar correctament.

- Connectar la placa MCP2210 via USB al ordinador. El programa mostrarà un missatge constant abaix a l'esquerra de la pantalla on s'hi observarà l'estat en el que es troba la connexió amb la MCP2210.
- Connectar la placa Nexys4 via USB al ordinador. Encendre la placa amb l'interruptor de ON/OFF que incorpora la placa. S'encendrà una llum vermella per demostrar que la placa està encesa.
- Connectar els cables de CLK, CS, MOSI i MISO de la següent manera:
    - CLK (PIN 7) → Port JC1
    - CS (PIN 1) → Port JC2 (en el cas de voler utilitzar GPIO 0)
    - MOSI (PIN 8) → Port JC3
    - MISO (PIN 6) → Port JC4

- Guia pas a pas

A continuació es mostraran tots els passos a seguir un a un per realitzar l'enviament de dades en direcció a la FPGA.

En qualssevol moment al llarg del procés d'enviament de dades es poden clicar els botons de **"Reset"** i **"Cerrar".**

- **"Reset":** Clicar aquest botó per esborrar totes les dades que s'hagin inclòs en el programa. Tant de la configuració SPI com de les dades a enviar.
- **"Cerrar":** Clicar aquest botó per tancar el programa.

1. Clicar en el programa per inicialitzar-lo.

Figura A1. Icona del programa.

2. Es mostrarà la pàgina de configuració de la connexió SPI.
   - Seleccionar un dels GP per on es transmetran les dades. Només se'n pot seleccionar un.
   - Introduir un **"bit-rate"**. És recomanable deixar-lo a 0 o en valors molt baixos per assegurar una bona comunicació SPI.
   - Un cop configurada la connexió clicar el botó **"Confirmar Configuración"** per accedir a la següent pestanya.

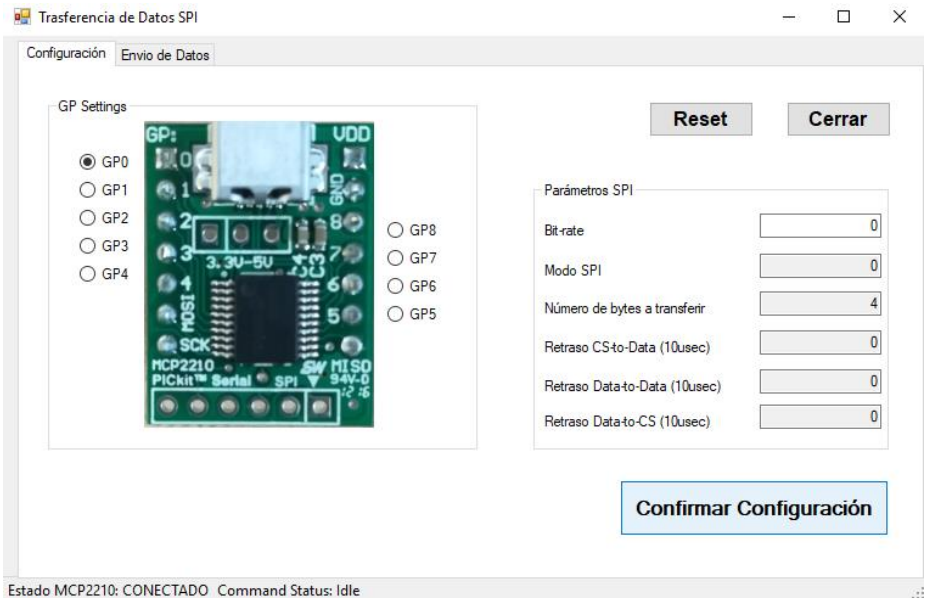

Figura A2. Pestanya de configuració.

3. Un cop premut el botó **"Confirmar Configuración"** apareixerà la pantalla d'introducció de dades. En aquesta pantalla es trobarà la gran majoria dels apartats bloquejats. Per desbloquejar-los haurà de seguir els següents passos:
   - Seleccionar **"Write"** per iniciar el procediment d'introducció de dades. Es desbloquejaran dues parts del programa: **"Introducción Automática"** i **"Introducción Manual"**.
   - Si es desitja utilitzar una de les plantilles ja programades que es proporcionen, es seleccionarà un dels nombres que es troben a **"Introducción Automática"**. Les dades

s'introduiran a **"Introducción Manual"** si es desitja modificar-les abans de la previsualització.

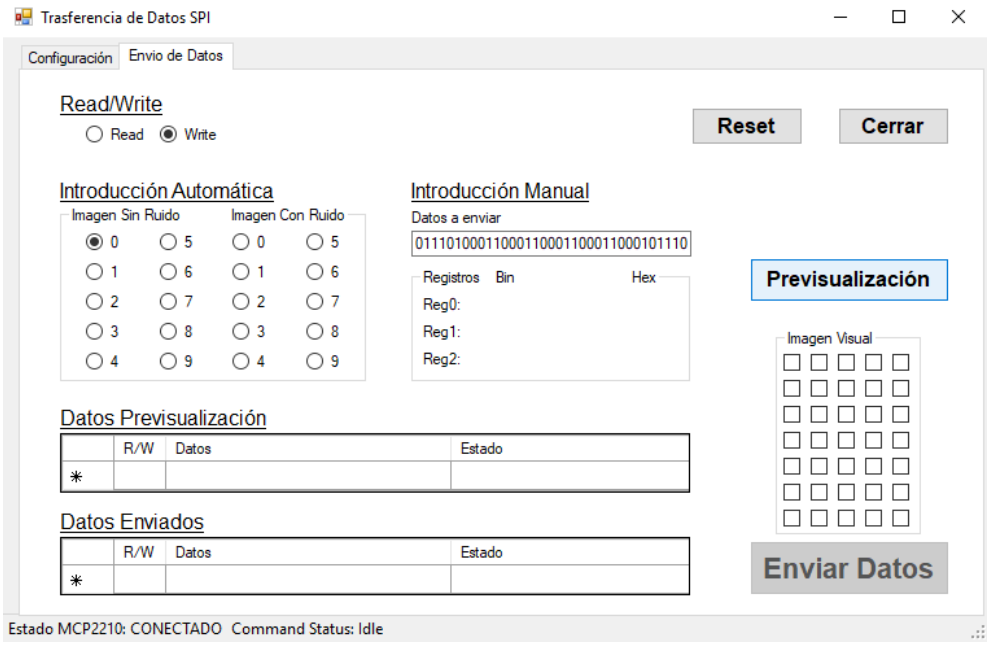

Figura A3. Pestanya d'enviament de dades abans de previsualitzar.

4. En el cas de que les dades siguin les adequades i es vulgui procedir, clicar el botó **"Previsualización"**. S'observarà com tant l'apartat **"Imagen Visual"**, **"Datos Previsualización"** i **"Registros"** s'actualitzen. Si la imatge apareix tal i com l'usuari ha sol·licitat es procedirà a l'enviament de dades.

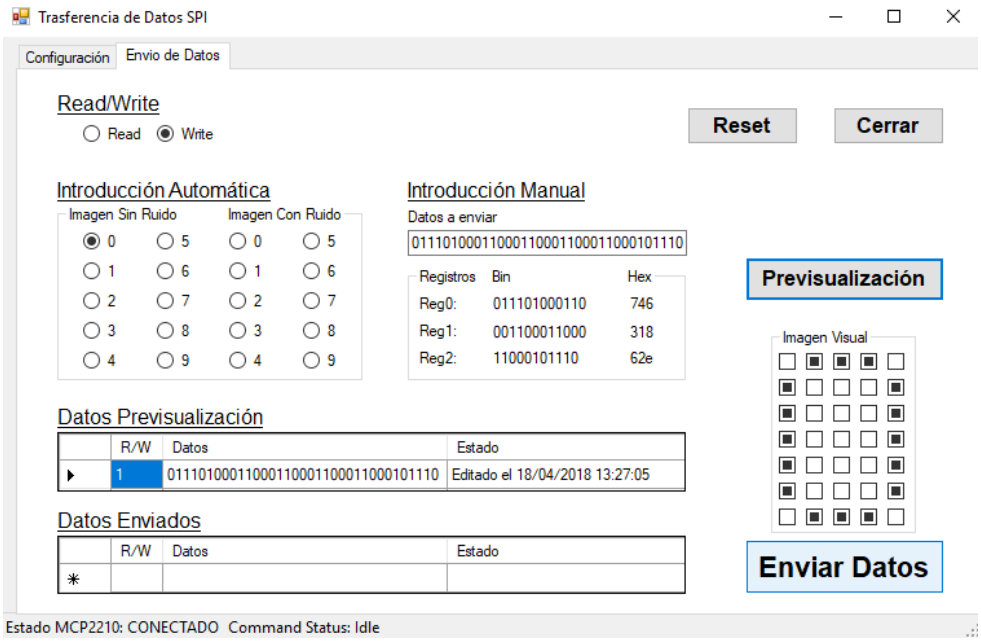

Figura A4. Pestanya d'enviament de dades després de previsualitzar.

5. Clicar el botó **"Enviar Datos"** per enviar les dades en direcció a la FPGA. Es podrà observar que les dades prèviament inserides a **"Datos Previsualización"** es traslladen a **"Datos Enviados"**, confirmant d'aquesta manera que les dades han estat emeses.

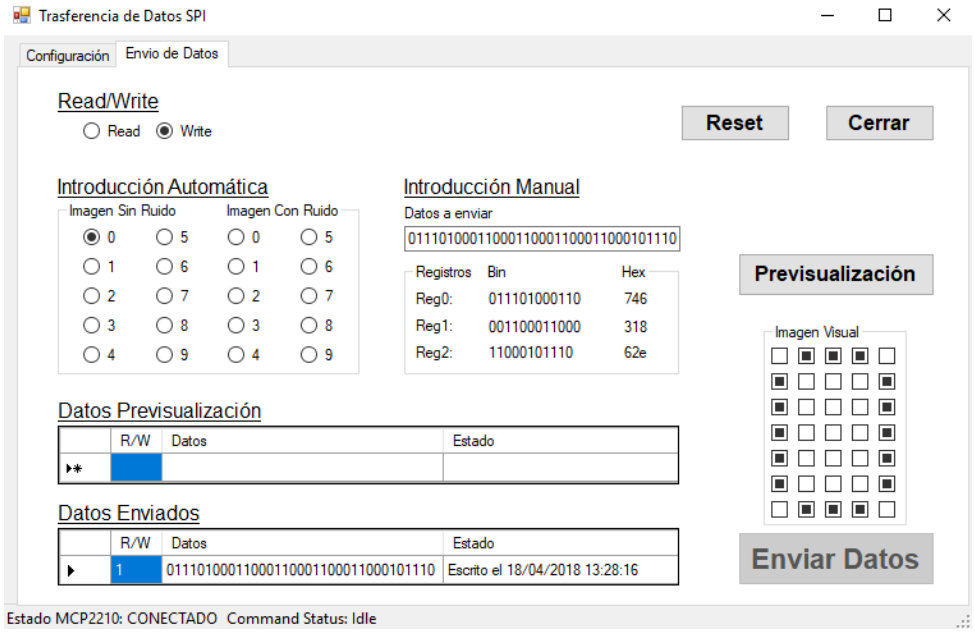

Figura A5. Pestanya d'enviament de dades un cop finalitzat el procés d'enviament.

A2. Codis programats

Codi VHDL

Slave SPI

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity spi_slave is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           spi_clk : in STD_LOGIC;
           spi_mosi : in STD_LOGIC;
           spi_cs : in STD_LOGIC;
           spi_miso : out STD_LOGIC;
           reg_out : out STD_LOGIC_VECTOR (15 downto 0);

           --7 segments
           seg: out std_logic_vector(6 downto 0);
           anode_activate: out std_logic_vector(7 downto 0);

           --test
           led : out std_logic_vector(15 downto 0);
           sw : in std_logic_vector(15 downto 0)
           );
end spi_slave;

architecture Behavioral of spi_slave is

component synchronizer
generic (
   longitud : integer := 2);
port (
   clk : in std_logic;
   rst : in std_logic;
   in_sig : in std_logic;
   out_sig : out std_logic);
end component;

--synchronizers
signal sclk,mosi,miso,cs : std_logic;
signal pre_sclk,flanc_sclk : std_logic;
signal registre : std_logic_vector (15 downto 0);

--7 segments
signal aux: std_logic_vector(11 downto 0);
signal LED_BCD: STD_LOGIC_VECTOR (3 downto 0);
signal refresh_counter: STD_LOGIC_VECTOR (20 downto 0);
-- the first 20-bit for creating refresh rate
signal LED_activating_counter: std_logic_vector(1 downto 0);
```

```
-- the other 2-bit for creating 4 LED-activating signals
-- count          0    ->  1  ->  2  ->  3
-- activates    LED1    LED2   LED3   LED4
-- and repeat

--test
signal reg0, reg1 : std_logic_vector(11 downto 0);
signal reg2 : std_logic_vector(10 downto 0);
signal write_out, read_out : std_logic_vector(34 downto 0);

type state_type is (idle, rx, transfer);
    signal state : state_type;

begin

    -- instanciacio dels sincronitzadors pels senyals spi
    --  l'objectiu es obtenir els mateixos senyals del bus spi
    --  pero sincronitzats amb el clk del sistema (clk)

    synch_SCLK : synchronizer -- sincronitzador spi_clk
       generic map (
          longitud => 2)
       port map (
          clk => clk,
          rst => rst,
          in_sig => spi_CLK,
          out_sig => sclk);

    synch_MOSI : synchronizer -- sincronitzador spi_MOSI
       generic map (
          longitud => 2)
       port map (
          clk => clk,
          rst => rst,
          in_sig => spi_MOSI,
          out_sig => mosi);

    synch_CS : synchronizer -- sincronitzador spi_CS
       generic map (
          longitud => 2)
       port map (
          clk => clk,
          rst => rst,
          in_sig => spi_CS,
          out_sig => cs);

    -- detecta flancs de sclk
    --    guarda el valor anterior de sclk a pre_sclk de manera
    --    que es podra comparar el valor anteior de sclk amb pre_sclk
    --    i aixi detectar quan ha passat de '1' a '0' (flan de baixada)
    process (clk)
    begin
        if (clk='1' and clk'event) then
            if (rst='1') then
                pre_sclk <= '0';
            else
                pre_sclk <= sclk;
            end if;
```

```
        end if;
    end process;

    -- activem flanc quan flanc de pujada de sclk i chip select (cs)
    flanc_sclk <= '1' when (sclk='1' and pre_sclk='0' and cs='0') else
'0';

    -- carrega entrada serie (MOSI) a registre de sortida de 16 bits
    --   fem sevir el senyal 'registre' com a vector temporal ja que al
ser 'reg_out'
    --   un vector de sortida no el podem llegir
    process (clk)
    begin
        if (clk='1' and clk'event) then
            if (rst = '1') then
                registre <= (others=>'0');
            elsif (flanc_sclk = '1') then
                registre(15 downto 1) <= registre(14 downto 0);
                registre(0) <= mosi;
            end if;
        end if;
    end process;

    maquina_estats: process(clk, rst)
    begin
         if (clk='1' and clk'event) then
            if (rst = '1') then
                state <= idle;
            else
                case state is
                    when idle =>
                        if spi_CS = '1' then
                            state <= idle;
                        else
                            state <= rx;
                        end if;
                    when rx =>
                        if spi_CS = '0' then
                            state <= rx;
                        else
                            state <= transfer;
                        end if;
                    when transfer =>
                        state <= idle;
                    when others =>
                        state <= idle;
                end case;
            end if;
        end if;
    end process;

    register_process: process(clk, rst)
    begin
         --s'emmagatzemen els valors segons l'adreça que
         --el missatge incorpori en els seus bits 14 i 13
         if (clk='1' and clk'event) then
            if (rst = '1') then
                reg0 <= (others=>'0');
```

```
                reg1 <= (others=>'0');
                reg2 <= (others=>'0');
            elsif (state = transfer) and (registre(15) = '1') then  --
registre(15) = '1' = write
                case (registre(14 downto 13)) is
                    when "00" => reg0 <= registre(12 downto 9) &
registre(7 downto 0);
                    when "01" => reg1 <= registre(12 downto 9) &
registre(7 downto 0);
                    when others => reg2 <= registre(12 downto 9) &
registre(6 downto 0);
                end case;
            end if;
        end if;
    end process;

    seven_segments_process:process(LED_BCD)
        begin
            --lògica negativa
           case LED_BCD is       --gfedcba
            when "0000" => seg <= "1000000"; -- "0"
            when "0001" => seg <= "1111001"; -- "1"
            when "0010" => seg <= "0100100"; -- "2"
            when "0011" => seg <= "0110000"; -- "3"
            when "0100" => seg <= "0011001"; -- "4"
            when "0101" => seg <= "0010010"; -- "5"
            when "0110" => seg <= "0000010"; -- "6"
            when "0111" => seg <= "1111000"; -- "7"
            when "1000" => seg <= "0000000"; -- "8"
            when "1001" => seg <= "0010000"; -- "9"
            when "1010" => seg <= "0100000"; -- a
            when "1011" => seg <= "0000011"; -- b
            when "1100" => seg <= "1000110"; -- C
            when "1101" => seg <= "0100001"; -- d
            when "1110" => seg <= "0000110"; -- E
            when "1111" => seg <= "0001110"; -- F

            when others => seg <= "0111111";
            end case;
        end process;

    --controlador del 7-segment display
    counter_process: process(clk,rst)
    begin
        if (clk'event and clk ='1') then
            if(rst='1') then
                refresh_counter <= (others => '0');
            else
                refresh_counter <= refresh_counter + 1;
            end if;
        end if;
    end process;

    LED_activating_counter <= refresh_counter(20 downto 19);
    -- 4-to-1 MUX per generar l'activació de l'ànode
    enable_process: process (clk, rst)--(LED_activating_counter)
    begin
        if (clk'event and clk = '1') then
```

```
            if (rst = '1') then
                anode_activate <= "01111000";
                LED_BCD <= "0000";
            else
            case LED_activating_counter is
                when "00" =>
                    anode_activate <= "11111011";
                    -- activate LED0
                    LED_BCD <= aux(11 downto 8);--"1100";
                when "01" =>
                    anode_activate <= "11111101";
                    -- activate LED1
                    LED_BCD <= aux(7 downto 4);--"1101";
                when "10" =>
                    anode_activate <= "11111110";
                    -- activate LED2
                    LED_BCD <= aux(3 downto 0);--"1110";
                when others =>
                    anode_activate <= "01111111";
                    -- activate LED8
                    LED_BCD <= "00" & bit_reg;
            end case;
          end if;
        end if;
    end process;

    visualitzacio_process: process(clk, rst)
    begin
        --segons els interruptors que s'activin es mostrarà
        --el registre 0, 1 o 2
        if(clk='1' and clk'event) then
            if (sw(0) = '0' and sw(1) = '0') then
                aux <= reg0;
                led <= reg0;
                bit_reg <= "00";
            elsif (sw(0) = '1' and sw(1) = '0') then
                aux <= reg1;
                led <= reg1;
                bit_reg <= "01";
            elsif (sw(0) = '0' and sw(1) = '1') then
                aux <= '0' & reg2;
                led <= '0' & reg2;
                bit_reg <= "10";
            else
                aux <= (others => '0');
                led <= (others => '0');
                bit_reg <= "11";
            end if;
        end if;
    end process;

    reg_out <= registre;
    write_out <= reg0 & reg1 & reg2;

end Behavioral;
```

Sincronitzador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity synchronizer is
    generic (longitud : integer := 2);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           in_sig : in STD_LOGIC;
           out_sig : out STD_LOGIC);
end synchronizer;

architecture Behavioral of synchronizer is

signal registre : std_logic_vector (longitud-1 downto 0);

begin

process(clk)
begin
    if (clk='1' and clk'event) then
        if (rst = '1') then
            registre <= (others => '0');
        else
            registre(longitud-1 downto 1) <= registre(longitud-2 downto
0);
            registre(0)<=in_sig;
        end if;
    end if;
end process;

out_sig <= registre(longitud - 1);

end Behavioral;
```

Constraints

```
set_property BITSTREAM.General.UnconstrainedPins {Allow} [current_design]
set_property SEVERITY {Warning} [get_drc_checks UCIO-1]
set_property SEVERITY {Warning} [get_drc_checks NSTD-1]

# Clock signal
#Bank = 35, Pin name = IO_L12P_T1_MRCC_35,           Sch name = CLK100MHZ
set_property PACKAGE_PIN E3 [get_ports {clk}]
    set_property IOSTANDARD LVCMOS33 [get_ports {clk}]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports {clk}]

set_property CFGBVS Vcco [current_design]
set_property config_voltage 3.3 [current_design]

### Switches
##Bank = 34, Pin name = IO_L21P_T3_DQS_34,                Sch name = SW0
set_property PACKAGE_PIN U9 [get_ports {sw[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
##Bank = 34, Pin name = IO_25_34,                         Sch name = SW1
set_property PACKAGE_PIN U8 [get_ports {sw[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
##Bank = 34, Pin name = IO_L23P_T3_34,                    Sch name = SW2
set_property PACKAGE_PIN R7 [get_ports {sw[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]

##Buttons
##Bank = 15, Pin name = IO_L11N_T1_SRCC_15,              Sch name = BTNC
set_property PACKAGE_PIN E16 [get_ports rst]
    set_property IOSTANDARD LVCMOS33 [get_ports rst]

##Pmod Header JC
#Bank = 35, Pin name = IO_L23P_T3_35,                    Sch name = JC1
set_property PACKAGE_PIN K2 [get_ports {spi_clk}]
    set_property IOSTANDARD LVCMOS33 [get_ports {spi_clk}]
#Bank = 35, Pin name = IO_L6P_T0_35,                     Sch name = JC2
set_property PACKAGE_PIN E7 [get_ports {spi_cs}]
    set_property IOSTANDARD LVCMOS33 [get_ports {spi_cs}]
#Bank = 35, Pin name = IO_L22P_T3_35,                    Sch name = JC3
set_property PACKAGE_PIN J3 [get_ports {spi_mosi}]
    set_property IOSTANDARD LVCMOS33 [get_ports {spi_mosi}]
#Bank = 35, Pin name = IO_L21P_T3_DQS_35,                Sch name = JC4
set_property PACKAGE_PIN J4 [get_ports {spi_miso}]
    set_property IOSTANDARD LVCMOS33 [get_ports {spi_miso}]


#7 segment display
    #Bank = 34, Pin name = IO_L2N_T0_34,                  Sch name = CA
    set_property PACKAGE_PIN L3 [get_ports {seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
    #Bank = 34, Pin name = IO_L3N_T0_DQS_34,              Sch name = CB
    set_property PACKAGE_PIN N1 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
    #Bank = 34, Pin name = IO_L6N_T0_VREF_34,             Sch name = CC
    set_property PACKAGE_PIN L5 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
```

```
#Bank = 34, Pin name = IO_L5N_T0_34,                    Sch name = CD
set_property PACKAGE_PIN L4 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
#Bank = 34, Pin name = IO_L2P_T0_34,                    Sch name = CE
set_property PACKAGE_PIN K3 [get_ports {seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
#Bank = 34, Pin name = IO_L4N_T0_34,                    Sch name = CF
set_property PACKAGE_PIN M2 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
#Bank = 34, Pin name = IO_L6P_T0_34,                    Sch name = CG
set_property PACKAGE_PIN L6 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]


#Bank = 34, Pin name = IO_L18N_T2_34,                   Sch name = AN0
set_property PACKAGE_PIN N6 [get_ports {anode_activate[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[0]}]
#Bank = 34, Pin name = IO_L18P_T2_34,                   Sch name = AN1
set_property PACKAGE_PIN M6 [get_ports {anode_activate[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[1]}]
#Bank = 34, Pin name = IO_L4P_T0_34,                    Sch name = AN2
set_property PACKAGE_PIN M3 [get_ports {anode_activate[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[2]}]
#Bank = 34, Pin name = IO_L13_T2_MRCC_34,               Sch name = AN3
set_property PACKAGE_PIN N5 [get_ports {anode_activate[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[3]}]
#Bank = 34, Pin name = IO_L3P_T0_DQS_34,                Sch name = AN4
set_property PACKAGE_PIN N2 [get_ports {anode_activate[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[4]}]
#Bank = 34, Pin name = IO_L16N_T2_34,                   Sch name = AN5
set_property PACKAGE_PIN N4 [get_ports {anode_activate[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[5]}]
#Bank = 34, Pin name = IO_L1P_T0_34,                    Sch name = AN6
set_property PACKAGE_PIN L1 [get_ports {anode_activate[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[6]}]
#Bank = 34, Pin name = IO_L1N_T034,                     Sch name = AN7
set_property PACKAGE_PIN M1 [get_ports {anode_activate[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {anode_activate[7]}]
```

Codi Visual Studio

Class Form_Terminal

```
Imports MCP2210
Imports MCP2210.DllConstants
Imports Microsoft.VisualBasic

'============================================================
Public Class Form_Terminal

    Dim MCP2210 As New ClassMCP2210

    Dim BytesStructure As New ClassDataStructure

    Dim FormControls As New ClassControls

    Public Const cNUM_GPs = 9

    Public bPinDes As Boolean
    Public bIdleCSVal As Boolean
    Public bActiveCSVal As Boolean

    Public ucPinDes(cNUM_GPs - 1), ucIdleCSVal(cNUM_GPs - 1),
    ucActiveCSVal(cNUM_GPs - 1) As Char

    '' SPI related
    Public TxData(65535 - 1), RxData(65535 - 1) As Byte
    Public paramBitRateValue As UInteger
    Public paramSpiMode As Byte
    Public paramNumBytes2Xfer As UInteger
    Public paramCS2DataDly As UInteger
    Public paramData2DataDly As UInteger
    Public paramData2CSDly As UInteger

    Public paramIdleCS As UInteger
    Public paramActiveCS As UInteger
    Public paramGPDesignation(cNUM_GPs - 1) As Byte


    '============================================================
    '============================================================

    Private Sub Form_Terminal_Load(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles MyBase.Load
        bPinDes = False
        bIdleCSVal = False
        bActiveCSVal = False

        'now call the UpdateGPValues()
        'UpdateGPValues()

        'set the number of visible rows to 1
        DataGridViewDatosEnviados.RowCount = 1
```

```
    'now enable the timer so we will update the APP status with the
    MCP2210 connection state
    Timer1.Enabled = True

End Sub

'============================================================
'============================================================
Private Sub ButtonCerrar_Click(sender As Object, e As EventArgs)
 Handles ButtonCerrar.Click, ButtonCerrar2.Click
    Me.Close()
End Sub

'============================================================
'============================================================
Private Sub ChangeTab(ByVal gp As Control)
    For Each rControl As Control In gp.Controls
        If rControl.GetType Is GetType(RadioButton) Then
            If DirectCast(rControl, RadioButton).Checked Then
                GroupBox_GPSettings.Enabled = False
                GroupBox_SPIParameters.Enabled = False
                TabControl1.SelectedTab = TabEnvioDatos
            End If
        End If
    Next
End Sub

'============================================================
'============================================================
Private Sub ButtonConfig_Click(sender As Object, e As EventArgs)
 Handles ButtonConfig.Click
    ChangeTab(GroupBox_GPSettings)
End Sub

'============================================================
'============================================================
Private Sub ResetAllControls(ByVal container As Control)
    For Each ctrl As Control In container.Controls
        If TypeOf ctrl Is RadioButton Then
            DirectCast(ctrl, RadioButton).Checked = False
        ElseIf TypeOf ctrl Is TextBox Then
            DirectCast(ctrl, TextBox).Clear()
        ElseIf TypeOf ctrl Is CheckBox Then
            DirectCast(ctrl, CheckBox).Checked = False
        ElseIf TypeOf ctrl Is DataGridView Then
            DirectCast(ctrl, DataGridView).Rows.Clear()
        End If
        If ctrl.Controls.Count > 0 Then
            ResetAllControls(ctrl)
        End If
    Next
End Sub

'============================================================
'============================================================
Private Sub Button_Reset_Click(ByVal sender As System.Object, ByVal e
 As System.EventArgs) Handles Button1.Click, ButtonReset2.Click
    ResetAllControls(TabEnvioDatos)
```

```
        ResetAllControls(GroupBox_GPSettings)
        TabControl1.SelectedTab = TabConfiguracion
        GroupBox_GPSettings.Enabled = True
        GroupBox_SPIParameters.Enabled = True
        Button_TxferSPIData.Enabled = False
        ButtonPrevisualizacion.Enabled = False
        LabelValue.Enabled = False
        TextBoxValue.Enabled = False
        GroupBoxSinRuido.Enabled = False
        LabelR0.Enabled = False
        LabelR1.Enabled = False
        LabelR2.Enabled = False
    End Sub

    '============================================================
    '============================================================
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
     System.EventArgs) Handles Timer1.Tick
        'use the timer callback to update the state of the MCP2210
     connection

        If (MCP2210.CheckUsbSpiConnection() <> True) Then
                    ToolStripStatusLabel_AppStatus.Text = "Estado MCP2210:
      NO CONECTADO"
        Else
                    ToolStripStatusLabel_AppStatus.Text = "Estado MCP2210:
      CONECTADO"
        End If
    End Sub
    '============================================================
    '============================================================
    Private Sub UpdateBytesForTransfer()
        Try
            BytesStructure.ListDataElements.Clear()

            For Each Row As DataGridViewRow In DataGridPreview.Rows
                If Not Row.IsNewRow Then

                    Dim Temporal As New ClassDataElement
                          Temporal.ReadWrite =
      Row.Cells("ColumnReadWrite").Value
                    Temporal.DataByte =
      Row.Cells("ColumnByteRegister").Value
                    Temporal.Status = Row.Cells("ColumnStatus").Value

                    BytesStructure.ListDataElements.Add(Temporal)
                End If
            Next

        Catch ex As Exception
                    MessageBox.Show("Hubo un error configurando los bytes a
      enviar.")
        End Try
    End Sub

    '============================================================
    '============================================================
    Private Sub UpdateGridAfterTransfer()
```

```
    Try
        If (RadioButtonRead.Checked = False) Then

            DataGridPreview.Rows.Clear()

  DataGridViewDatosEnviados.Rows.Add(BytesStructure.ListDataElements.
  Count)

            For i As Integer = 0 To
  BytesStructure.ListDataElements.Count - 1
                ' Update the value of each of the columns with the
  information fo the object parameters
                Dim Element As ClassDataElement =
  BytesStructure.ListDataElements(i)
                Dim Row As DataGridViewRow =
  DataGridViewDatosEnviados.Rows(i)
                Row.Cells("ColumnReadWrite1").Value =
  Element.ReadWrite
                Row.Cells("ColumnByteRegister1").Value =
  Element.DataByte
                Row.Cells("ColumnStatus1").Value = Element.Status
            Next
        Else
            DataGridPreview.Rows.Clear()
        End If

    Catch ex As Exception
              MessageBox.Show("Hubo un error al mostrar los datos de
  la transferencia.")
    End Try
End Sub
'=============================================================
'=============================================================
Private Sub Button_TxferSPIData_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles Button_TxferSPIData.Click
    ' Updates the SPI Parameters sending the information of the
 Configuration Window
    MCP2210.UpdateSPIParameters(Me.TextBox_SpiMode.Text,

  Me.TextBox_BitRate.Text,

  Me.TextBox_CS2DataDly.Text,

  Me.TextBox_Data2DataDly.Text,

  Me.TextBox_Data2CSDly.Text)


    ' Updates the bytes for transfer and after that executes the
  communication
    UpdateBytesForTransfer()
    MCP2210.TransferSpiData(BytesStructure)

    ' When the communication is finished updates the grid and
  controls
    UpdateGridAfterTransfer()
    Button_TxferSPIData.Enabled = False
End Sub
```

```
'=============================================================
'=============================================================
Private Sub DataGridView_TxData_CellEndEdit(ByVal sender As
 System.Object, ByVal e As
 System.Windows.Forms.DataGridViewCellEventArgs) Handles
 DataGridViewDatosEnviados.CellEndEdit
    Dim data_cell As DataGridViewCell
    Dim data_row As DataGridViewRow
    Dim strCellVal As String
    Dim iCellVal As Integer

    data_row = DataGridViewDatosEnviados.Rows.Item(e.RowIndex)
    data_cell = data_row.Cells.Item(e.ColumnIndex)

    strCellVal = data_cell.Value

    Try
        iCellVal = System.Convert.ToByte(strCellVal, 16)
    Catch ex As Exception
        'keep the old value - do nothing
        iCellVal = 0
    End Try

    TxData(e.RowIndex) = iCellVal
    data_cell.Value = iCellVal.ToString("X2")

End Sub
'=============================================================
Public Sub UpdateSPIParameters()
    'update the SPI textboxes
    TextBox_BitRate.Text = paramBitRateValue
    TextBox_SpiMode.Text = paramSpiMode
    TextBox_NumBytes2Xfer.Text = paramNumBytes2Xfer
    TextBox_CS2DataDly.Text = paramCS2DataDly
    TextBox_Data2DataDly.Text = paramData2DataDly
    TextBox_Data2CSDly.Text = paramData2CSDly
End Sub
'=============================================================
'=============================================================
Public Function GetSpiParameters() As Boolean
    'convert the bitrate
    Try
              paramBitRateValue =
  System.Convert.ToUInt32(TextBox_BitRate.Text, 10)
    Catch ex As Exception
        'abort the operation
              ToolStripStatusLabel_CmdStatus.Text = "Command Status:
  BitRate value wrong"
        Return False
    End Try
    'convert the SpiMode
    Try
              paramSpiMode =
  System.Convert.ToUInt32(TextBox_SpiMode.Text, 10)
    Catch ex As Exception
        'abort the operation
```

```
            ToolStripStatusLabel_CmdStatus.Text = "Command Status:
Spi Mode value wrong"
      Return False
  End Try
  'convert the NumBytes2Xfer
  Try
            paramNumBytes2Xfer =
System.Convert.ToUInt32(TextBox_NumBytes2Xfer.Text, 10)
  Catch ex As Exception
      'abort the operation
            ToolStripStatusLabel_CmdStatus.Text = "Command Status:
Number of bytes for transfer value wrong"
      Return False
  End Try
  'convert the CS2DataDly
  Try
            paramCS2DataDly =
System.Convert.ToUInt32(TextBox_CS2DataDly.Text, 10)
  Catch ex As Exception
      'abort the operation
            ToolStripStatusLabel_CmdStatus.Text = "Command Status:
CS to Data delay value wrong"
      Return False
  End Try
  'convert the Data2DataDly
  Try
            paramData2DataDly =
System.Convert.ToUInt32(TextBox_Data2DataDly.Text, 10)
  Catch ex As Exception
      'abort the operation
            ToolStripStatusLabel_CmdStatus.Text = "Command Status:
Data to Data delay value wrong"
      Return False
  End Try
  'convert the Data2CSDly
  Try
            paramData2CSDly =
System.Convert.ToUInt32(TextBox_Data2CSDly.Text, 10)
  Catch ex As Exception
      'abort the operation
            ToolStripStatusLabel_CmdStatus.Text = "Command Status:
Data to CS delay value wrong"
      Return False
  End Try
  Dim strValue As String

  'convert the IDLE CS value
  strValue = CStr(ucIdleCSVal)
  Try
      paramIdleCS = System.Convert.ToUInt16(strValue, 2)
  Catch ex As Exception
      'abort the operation
            ToolStripStatusLabel_CmdStatus.Text = "Command Status:
IDLE CS delay value wrong"
      Return False
  End Try
  'convert the ACTIVE CS value
  strValue = CStr(ucActiveCSVal)
```

```
        Try
            paramActiveCS = System.Convert.ToUInt16(strValue, 2)
        Catch ex As Exception
            'abort the operation
                  ToolStripStatusLabel_CmdStatus.Text = "Command Status:
  ACTIVE CS delay value wrong"
            Return False
        End Try
        'check if we need to fill the SPI TX user data
        Dim ucFillValue As Byte

        If (CheckBox_FillData.Checked = True) Then
                  For counter = (DataGridViewDatosEnviados.Rows.Count -
  1) To (paramNumBytes2Xfer - 1) Step 1
              TxData(counter) = ucFillValue
            Next
        End If
        'everything went well
        Return True
    End Function
    '=============================================================
     '=============================================================
    Private Sub RadioButtonSR0_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonSR0.CheckedChanged
        If RadioButtonSR0.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "01110100011000110001100011000101110"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonSR1_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonSR1.CheckedChanged
        If RadioButtonSR1.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "00100011000010000100001000010001110"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonSR2_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonSR2.CheckedChanged
        If RadioButtonSR2.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "01110100010000100010001000100011111"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonSR3_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonSR3.CheckedChanged
        If RadioButtonSR3.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "01110100010000100110000011000101110"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonSR4_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonSR4.CheckedChanged
        If RadioButtonSR4.Checked = True Then
            TextBoxValue.Clear()
```

```
        TextBoxValue.Text = "00010001100101010010111110001000010"
    End If
End Sub
'=============================================================
Private Sub RadioButtonSR5_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonSR5.CheckedChanged
    If RadioButtonSR5.Checked = True Then
        TextBoxValue.Clear()
        TextBoxValue.Text = "11111100001111000001000011000101110"
    End If
End Sub
'=============================================================
Private Sub RadioButtonSR6_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonSR6.CheckedChanged
    If RadioButtonSR6.Checked = True Then
        TextBoxValue.Clear()
        TextBoxValue.Text = "00110010001000011110100011000101110"
    End If
End Sub
'=============================================================
Private Sub RadioButtonSR7_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonSR7.CheckedChanged
    If RadioButtonSR7.Checked = True Then
        TextBoxValue.Clear()
        TextBoxValue.Text = "11111000010001000100010000100001000"
    End If
End Sub
'=============================================================
Private Sub RadioButtonSR8_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonSR8.CheckedChanged
    If RadioButtonSR8.Checked = True Then
        TextBoxValue.Clear()
        TextBoxValue.Text = "01110100011000101110100011000101110"
    End If
End Sub
'=============================================================
Private Sub RadioButtonSR9_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonSR9.CheckedChanged
    If RadioButtonSR9.Checked = True Then
        TextBoxValue.Clear()
        TextBoxValue.Text = "01110100011000101111000010001001100"
    End If
End Sub
'=============================================================
'=============================================================
'=============================================================
Private Sub RadioButtonCR0_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonCR0.CheckedChanged
    If RadioButtonCR0.Checked = True Then
        TextBoxValue.Clear()
        TextBoxValue.Text = "11111100011000110001100011000111111"
    End If
End Sub
'=============================================================
Private Sub RadioButtonCR1_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonCR1.CheckedChanged
    If RadioButtonCR1.Checked = True Then
        TextBoxValue.Clear()
```

```
            TextBoxValue.Text = "00100011000010000100001000010000100"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonCR2_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonCR2.CheckedChanged
        If RadioButtonCR2.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "01110000010000100010001000100001111"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonCR3_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonCR3.CheckedChanged
        If RadioButtonCR3.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "01110000010000100110000010000101110"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonCR4_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonCR4.CheckedChanged
        If RadioButtonCR4.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "00010100101001010010111110001000010"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonCR5_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonCR5.CheckedChanged
        If RadioButtonCR5.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "11110100001111000001000010000101110"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonCR6_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonCR6.CheckedChanged
        If RadioButtonCR6.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "00110010001000001110100001000101110"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonCR7_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonCR7.CheckedChanged
        If RadioButtonCR7.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "11111000010001000000010000100001000"
        End If
    End Sub
    '=============================================================
    Private Sub RadioButtonCR8_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButtonCR8.CheckedChanged
        If RadioButtonCR8.Checked = True Then
            TextBoxValue.Clear()
            TextBoxValue.Text = "01110100011000111111100011000101110"
        End If
```

```
End Sub
'=============================================================
Private Sub RadioButtonCR9_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonCR9.CheckedChanged
    If RadioButtonCR9.Checked = True Then
        TextBoxValue.Clear()
        TextBoxValue.Text = "01110100011000101110000010001000100"
    End If
End Sub

'=============================================================
'=============================================================
Private Sub RadioButtonRead_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonRead.CheckedChanged
    LabelValue.Enabled = False
    TextBoxValue.Enabled = False
    TextBoxValue.Text = ""
    GroupBoxSinRuido.Enabled = False
    ButtonPrevisualizacion.Enabled = True
    ResetAllControls(GroupBoxSinRuido)
End Sub
'=============================================================
'=============================================================
Private Sub RadioButtonWrite_CheckedChanged(sender As Object, e As
 EventArgs) Handles RadioButtonWrite.CheckedChanged
    LabelValue.Enabled = True
    TextBoxValue.Enabled = True
    GroupBoxSinRuido.Enabled = True
    LabelR0.Enabled = True
    LabelR1.Enabled = True
    LabelR2.Enabled = True
    GroupBoxRegistros.Enabled = True
    ButtonPrevisualizacion.Enabled = True

End Sub
'=============================================================
'=============================================================
Private Sub ButtonPrevisualizacion_Click(sender As Object, e As
 EventArgs) Handles ButtonPrevisualizacion.Click
    Try
        FormControls.UpdateData()

        If RadioButtonWrite.Checked = True Then
            LabelReg0.Text = TextBoxValue.Text.Substring(0, 12)
            LabelReg1.Text = TextBoxValue.Text.Substring(12, 12)
            LabelReg2.Text = TextBoxValue.Text.Substring(24, 11)

            LabelReg0Hex.Text =
  Convert.ToString(Convert.ToInt32(TextBoxValue.Text.Substring(0,
  12), 2), 16)
            LabelReg1Hex.Text =
  Convert.ToString(Convert.ToInt32(TextBoxValue.Text.Substring(12,
  12), 2), 16)
            LabelReg2Hex.Text =
  Convert.ToString(Convert.ToInt32(TextBoxValue.Text.Substring(24,
  11), 2), 16)
        End If
```

```
Button_TxferSPIData.Enabled = True

If RadioButtonWrite.Checked = True Then

    If TextBoxValue.Text(0) = "1" Then
        CheckBox0x0.CheckState = CheckState.Indeterminate
    Else
        CheckBox0x0.CheckState = CheckState.Unchecked
    End If

    If TextBoxValue.Text(1) = "1" Then
        CheckBox0x1.CheckState = CheckState.Indeterminate
    Else
        CheckBox0x1.CheckState = CheckState.Unchecked
    End If

    If TextBoxValue.Text(2) = "1" Then
        CheckBox0x2.CheckState = CheckState.Indeterminate
    Else
        CheckBox0x2.CheckState = CheckState.Unchecked
    End If

    If TextBoxValue.Text(3) = "1" Then
        CheckBox0x3.CheckState = CheckState.Indeterminate
    Else
        CheckBox0x3.CheckState = CheckState.Unchecked
    End If

    If TextBoxValue.Text(4) = "1" Then
        CheckBox0x4.CheckState = CheckState.Indeterminate
    Else
        CheckBox0x4.CheckState = CheckState.Unchecked
    End If

    If TextBoxValue.Text(5) = "1" Then
        CheckBox1x0.CheckState = CheckState.Indeterminate
    Else
        CheckBox1x0.CheckState = CheckState.Unchecked
    End If

    If TextBoxValue.Text(6) = "1" Then
        CheckBox1x1.CheckState = CheckState.Indeterminate
    Else
        CheckBox1x1.CheckState = CheckState.Unchecked
    End If

    If TextBoxValue.Text(7) = "1" Then
        CheckBox1x2.CheckState = CheckState.Indeterminate
    Else
        CheckBox1x2.CheckState = CheckState.Unchecked
    End If

    If TextBoxValue.Text(8) = "1" Then
        CheckBox1x3.CheckState = CheckState.Indeterminate
    Else
        CheckBox1x3.CheckState = CheckState.Unchecked
    End If
```

```
If TextBoxValue.Text(9) = "1" Then
    CheckBox1x4.CheckState = CheckState.Indeterminate
Else
    CheckBox1x4.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(10) = "1" Then
    CheckBox2x0.CheckState = CheckState.Indeterminate
Else
    CheckBox2x0.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(11) = "1" Then
    CheckBox2x1.CheckState = CheckState.Indeterminate
Else
    CheckBox2x1.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(12) = "1" Then
    CheckBox2x2.CheckState = CheckState.Indeterminate
Else
    CheckBox2x2.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(13) = "1" Then
    CheckBox2x3.CheckState = CheckState.Indeterminate
Else
    CheckBox2x3.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(14) = "1" Then
    CheckBox2x4.CheckState = CheckState.Indeterminate
Else
    CheckBox2x4.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(15) = "1" Then
    CheckBox3x0.CheckState = CheckState.Indeterminate
Else
    CheckBox3x0.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(16) = "1" Then
    CheckBox3x1.CheckState = CheckState.Indeterminate
Else
    CheckBox3x1.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(17) = "1" Then
    CheckBox3x2.CheckState = CheckState.Indeterminate
Else
    CheckBox3x2.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(18) = "1" Then
    CheckBox3x3.CheckState = CheckState.Indeterminate
Else
    CheckBox3x3.CheckState = CheckState.Unchecked
```

```
End If

If TextBoxValue.Text(19) = "1" Then
    CheckBox3x4.CheckState = CheckState.Indeterminate
Else
    CheckBox3x4.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(20) = "1" Then
    CheckBox4x0.CheckState = CheckState.Indeterminate
Else
    CheckBox4x0.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(21) = "1" Then
    CheckBox4x1.CheckState = CheckState.Indeterminate
Else
    CheckBox4x1.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(22) = "1" Then
    CheckBox4x2.CheckState = CheckState.Indeterminate
Else
    CheckBox4x2.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(23) = "1" Then
    CheckBox4x3.CheckState = CheckState.Indeterminate
Else
    CheckBox4x3.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(24) = "1" Then
    CheckBox4x4.CheckState = CheckState.Indeterminate
Else
    CheckBox4x4.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(25) = "1" Then
    CheckBox5x0.CheckState = CheckState.Indeterminate
Else
    CheckBox5x0.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(26) = "1" Then
    CheckBox5x1.CheckState = CheckState.Indeterminate
Else
    CheckBox5x1.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(27) = "1" Then
    CheckBox5x2.CheckState = CheckState.Indeterminate
Else
    CheckBox5x2.CheckState = CheckState.Unchecked
End If

If TextBoxValue.Text(28) = "1" Then
    CheckBox5x3.CheckState = CheckState.Indeterminate
```

```
                Else
                    CheckBox5x3.CheckState = CheckState.Unchecked
                End If

                If TextBoxValue.Text(29) = "1" Then
                    CheckBox5x4.CheckState = CheckState.Indeterminate
                Else
                    CheckBox5x4.CheckState = CheckState.Unchecked
                End If

                If TextBoxValue.Text(30) = "1" Then
                    CheckBox6x0.CheckState = CheckState.Indeterminate
                Else
                    CheckBox6x0.CheckState = CheckState.Unchecked
                End If

                If TextBoxValue.Text(31) = "1" Then
                    CheckBox6x1.CheckState = CheckState.Indeterminate
                Else
                    CheckBox6x1.CheckState = CheckState.Unchecked
                End If

                If TextBoxValue.Text(32) = "1" Then
                    CheckBox6x2.CheckState = CheckState.Indeterminate
                Else
                    CheckBox6x2.CheckState = CheckState.Unchecked
                End If

                If TextBoxValue.Text(33) = "1" Then
                    CheckBox6x3.CheckState = CheckState.Indeterminate
                Else
                    CheckBox6x3.CheckState = CheckState.Unchecked
                End If

                If TextBoxValue.Text(34) = "1" Then
                    CheckBox6x4.CheckState = CheckState.Indeterminate
                Else
                    CheckBox6x4.CheckState = CheckState.Unchecked
                End If
            End If


        Catch ex As Exception

        End Try
    End Sub

End Class
```

Class Form_Terminal

```
Public Class ClassMCP2210

    Private Const VID = &H4D8
    Private Const PID = &HDE
    Private Const cNUM_GPs = 9
    Private UsbSpi As New MCP2210.DevIO(VID, PID)
    Private bMCP2210Connection As Boolean


    Private bTxData(5) As Byte

    Public Property TxData() As Byte()
        Get
            Return bTxData
        End Get
        Set(ByVal value As Byte())
            bTxData = value
        End Set
    End Property

    Private bRxData(5) As Byte

    Public Property RxData() As Byte()
        Get
            Return bRxData
        End Get
        Set(ByVal value As Byte())
            bRxData = value
        End Set
    End Property

    Private iparamNumBytes2Xfer As UInteger

    Public Property paramNumBytes2Xfer() As UInteger
        Get
            Return iparamNumBytes2Xfer
        End Get
        Set(ByVal value As UInteger)
            iparamNumBytes2Xfer = value
        End Set
    End Property

    Private aparamGPDesignation(cNUM_GPs - 1) As Byte

    Public Property paramGPDesignation() As Byte()
        Get
            Return aparamGPDesignation
        End Get
        Set(ByVal value As Byte())
            aparamGPDesignation = value
        End Set
    End Property

    Private paramBitRateValue As UInteger = 1464
    Private paramSpiMode As Byte = 0
```

```
Private paramCS2DataDly As UInteger = 0
Private paramData2DataDly As UInteger = 0
Private paramData2CSDly As UInteger = 0
Private paramIdleCS As UInteger = 511
Private paramActiveCS As UInteger = 0

Private DigitalConv As New ClassDigitalConversion
      Public Sub UpdateSPIParameters(ByVal SPImode As Integer, ByVal
  BitRate As UInteger,
                                ByVal CsToDataDelay As UInteger, ByVal
  DataToDataDelay As UInteger,
                                ByVal DataToCsDelay As UInteger)
    paramNumBytes2Xfer = 2
    paramGPDesignation(0) = 1
    paramGPDesignation(1) = 1
    paramGPDesignation(2) = 1
    paramGPDesignation(3) = 1
    paramGPDesignation(4) = 1
    paramGPDesignation(5) = 1
    paramGPDesignation(6) = 1
    paramGPDesignation(7) = 1
    paramGPDesignation(8) = 1
    paramSpiMode = SPImode
    paramBitRateValue = BitRate
    paramCS2DataDly = CsToDataDelay
    paramData2DataDly = DataToDataDelay
    paramData2CSDly = DataToCsDelay
    'paramActiveCS = Convert.ToUInt16(EnablesString, 2)
End Sub


Function CheckUsbSpiConnection() As Boolean
    bMCP2210Connection = UsbSpi.Settings.GetConnectionStatus()
    Return bMCP2210Connection
End Function

'Transers the spi data from the information to a ist of elements.

''' <param name="DataList">List of elements with the information to
transfer.</param>

Public Sub TransferSpiData(ByVal dataList As ClassDataStructure)

    If CheckUsbSpiConnection() Then

        dataList.UpdateBytesParameters()

        For Each Element As ClassDataElement In
  dataList.ListDataElements

            Dim TextBoxValue As TextBox = Form_Terminal.TextBoxValue
                  Dim RadioButtonRead As RadioButton =
  Form_Terminal.RadioButtonRead
            Dim RadioButtonWrite As RadioButton =
  Form_Terminal.RadioButtonWrite
            Dim ReadWrite As String

            If RadioButtonRead.Checked Then
```

```
                ReadWrite = 0
            ElseIf RadioButtonWrite.Checked Then
                ReadWrite = 1
            End If

            Dim Auxiliar0 As String
            If RadioButtonWrite.Checked = True Then
                    Auxiliar0 = ReadWrite + "00" +
  TextBoxValue.Text.Substring(0, 4) + "0"
                TxData(0) = Convert.ToByte(Auxiliar0, 2)
                    TxData(1) =
  Convert.ToByte(TextBoxValue.Text.Substring(4, 8), 2)
                CommunicationExecution()
                Threading.Thread.Sleep(10)

                Auxiliar0 = ReadWrite + "01" +
  TextBoxValue.Text.Substring(12, 4) + "0"
                TxData(0) = Convert.ToByte(Auxiliar0, 2)
                    TxData(1) =
  Convert.ToByte(TextBoxValue.Text.Substring(16, 8), 2)
                CommunicationExecution()
                Threading.Thread.Sleep(10)

                Auxiliar0 = ReadWrite + "10" +
  TextBoxValue.Text.Substring(24, 4) + "0"
                TxData(0) = Convert.ToByte(Auxiliar0, 2)
                    TxData(1) =
  Convert.ToByte(TextBoxValue.Text.Substring(28, 7), 2)
                CommunicationExecution()
                Threading.Thread.Sleep(10)
            End If

            'update status
            If Element.ReadWrite = "0" Then
                Element.Status = "Leído el " + Date.Now.ToString
            Else
                Element.Status = "Escrito el " + Date.Now.ToString
            End If
            Threading.Thread.Sleep(50)

            'exits the loop once the status is updated
            Exit For
        Next
    Else
        MessageBox.Show("MCP2210 NO CONECTADO!")
    End If
End Sub


Private Sub CommunicationExecution()
    Form_Terminal.Timer1.Stop()
    If CheckUsbSpiConnection() Then
        Dim iResult As Integer
        'set the GP designations
              iResult =
  UsbSpi.Settings.SetGpioConfig(MCP2210.DllConstants.CURRENT_SETTINGS
  _ONLY, paramGPDesignation, &HFFFF, &HFFFF)
```

```
        If iResult <> 0 Then
            Console.WriteLine("SetGpioConfig failed")
            Return
        End If

        'now that we have the SPI parameters, we will setup the chip
              iResult =
  UsbSpi.Settings.SetAllSpiSettings(MCP2210.DllConstants.CURRENT_SETT
  INGS_ONLY,

        paramBitRateValue,
        paramIdleCS,
        paramActiveCS,
        paramCS2DataDly,
        paramData2DataDly,
        paramData2CSDly,
        paramNumBytes2Xfer,
        paramSpiMode)

        If iResult <> 0 Then
            Console.WriteLine("SetAllSpiSettings failed")
            Return
        End If

        'now call the SPI transfer function
        iResult = UsbSpi.Functions.TxferSpiData(TxData, RxData)

        If iResult <> 0 Then
            Console.WriteLine("TxferSpiData failed")
            Return
        End If

        'get the current chip settings
              paramBitRateValue =
  UsbSpi.Settings.GetSpiBitRate(MCP2210.DllConstants.CURRENT_SETTINGS
  _ONLY)
        paramSpiMode =
  UsbSpi.Settings.GetSpiMode(MCP2210.DllConstants.CURRENT_SETTINGS_ON
  LY)
        paramNumBytes2Xfer =
  UsbSpi.Settings.GetSpiTxferSize(MCP2210.DllConstants.CURRENT_SETTIN
  GS_ONLY)
        paramCS2DataDly =
  UsbSpi.Settings.GetSpiDelayCsToData(MCP2210.DllConstants.CURRENT_SE
  TTINGS_ONLY)
        paramData2DataDly =
  UsbSpi.Settings.GetSpiDelayDataToData(MCP2210.DllConstants.CURRENT_
  SETTINGS_ONLY)
        paramData2CSDly =
  UsbSpi.Settings.GetSpiDelayDataToCs(MCP2210.DllConstants.CURRENT_SE
  TTINGS_ONLY)
        Form_Terminal.Timer1.Start()
    Else
        Form_Terminal.Timer1.Start()
        Return
    End If
    Form_Terminal.Timer1.Start()
End Sub
```

```
End Class
```

Class ClassDataStructure

```
Public Class ClassDataStructure

    Private lDataElements As New List(Of ClassDataElement)
    ''' <summary>
    ''' List of ClassDataElement
    ''' </summary>
          ''' <value>List of ClassDataElement, that contains all
      the information of each element</value>
    ''' <remarks></remarks>
    Public Property ListDataElements() As List(Of ClassDataElement)
        Get
            Return lDataElements
        End Get
        Set(ByVal value As List(Of ClassDataElement))
            lDataElements = value
        End Set
    End Property

    ''' <summary>
    ''' Updates the bytes parameters from the string address
    ''' </summary>
          ''' <remarks>Needs that all the elements have the string
      address defined.</remarks>
    Public Sub UpdateBytesParameters()
        ' Loop through all the elments in the list
        For Each Element As ClassDataElement In ListDataElements

        Next
    End Sub
End Class
```

Class ClassDataStructure

```
Public Class ClassControls

    'list of controls for each element
    Private lListFormElements As New List(Of ClassControlsInfo)

    Public Property ListFormElements() As List(Of ClassControlsInfo)
        Get
            Return lListFormElements
        End Get
        Set(ByVal value As List(Of ClassControlsInfo))
            lListFormElements = value
        End Set
    End Property

    Private DigitalConv As New ClassDigitalConversion

    Public Function UpdateData() As Boolean
        Try
            Dim iAddress As Integer = 0
            Dim DataGridPreview As DataGridView =
      Form_Terminal.DataGridPreview
            DataGridPreview.Rows.Clear()
            DataGridPreview.Rows.Add(1)

            For i As Integer = 0 To 1 Step 1

                Dim RadioButtonRead As RadioButton =
      Form_Terminal.RadioButtonRead
                Dim RadioButtonWrite As RadioButton =
      Form_Terminal.RadioButtonWrite
                Dim TextBoxValue As TextBox = Form_Terminal.TextBoxValue

                DataGridPreview.Rows.Add()

                Dim ElementRow As DataGridViewRow =
      DataGridPreview.Rows(i)

                If RadioButtonRead.Checked = False And
      RadioButtonWrite.Checked = False Then
                    'none checked Read/Write, show error message
                    MessageBox.Show("Datos no vàlidos. Seleccione
      Read/Write.")
                    DataGridPreview.Rows.Clear()
                    Return False
                ElseIf RadioButtonRead.Checked = True Then
                    ElementRow.Cells("ColumnReadWrite").Value = 0
                Else
                    ElementRow.Cells("ColumnReadWrite").Value = 1
                End If

                If TextBoxValue.Text = "" And RadioButtonWrite.Checked =
      True Then
                    MessageBox.Show("Datos no vàlidos. Introduzca el
      valor de los datos a enviar.")
                    DataGridPreview.Rows.Clear()
```

```
                    Return False
                End If

                ElementRow.Cells("ColumnByteRegister").Value =
      TextBoxValue.Text

                ElementRow.Cells("ColumnStatus").Value = "Editado el " +
      Date.Now.ToString
            Next
            Return True

        Catch ex As Exception
            Return False
            Throw New Exception(ex.ToString)
        End Try
    End Function

    Public Sub UpdateAfterTransfer()
        For i As Integer = 0 To ListFormElements.Count - 1
            Dim GridRow As DataGridViewRow =
     Form_Terminal.DataGridPreview.Rows(i)
            Dim iString As String = i.ToString("00")
        Next
    End Sub
End Class
```